UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

Um Esquema Rápido Baseado em Aprendizado de Máquina para a Predição Interquadros do Codificador de Vídeo VVC

Paulo Henrik Ribeiro Gonçalves

Pelotas, 2021

Paulo Henrik Ribeiro Gonçalves

Um Esquema Rápido Baseado em Aprendizado de Máquina para a Predição Interquadros do Codificador de Vídeo VVC

Dissertação apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Schiavon Porto
Coorientadores: Prof. Dr. Guilherme Ribeiro Corrêa
Prof. Dr. Luciano Volcan Agostini

Pelotas, 2021

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

G635e Gonçalves, Paulo Henrik Ribeiro

Um esquema rápido baseado em aprendizado de máquina para a predição interquadros do codificador de vídeo VVC / Paulo Henrik Ribeiro Gonçalves ; Marcelo Schiavon Porto, orientador ; Guilherme Ribeiro Corrêa, Luciano Volcan Agostini, coorientadores. — Pelotas, 2021.

90 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2021.

1. Codificação de vídeo. 2. Aprendizado de máquina. 3. Redução de esforço computacional. 4. Predição affine. I. Porto, Marcelo Schiavon, orient. II. Corrêa, Guilherme Ribeiro, coorient. III. Agostini, Luciano Volcan, coorient. IV. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Paulo Henrik Ribeiro Gonçalves

Um Esquema Rápido Baseado em Aprendizado de Máquina para a Predição Interquadros do Codificador de Vídeo VVC

Dissertação aprovada, como requisito parcial, para obtenção do grau de Mestre em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 23 de Abril de 2021

Banca Examinadora:

Prof. Dr. Marcelo Schiavon Porto (orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Marilton Sanchotene de Aguiar
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Mateus Grellert da Silva
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Gustavo Freitas Sanchez
Doutor em Computação pela Pontifícia Universidade Católica do Rio Grande do Sul.

RESUMO

GONÇALVES, Paulo Henrik Ribeiro. **Um Esquema Rápido Baseado em Aprendizado de Máquina para a Predição Interquadros do Codificador de Vídeo VVC**. Orientador: Marcelo Schiavon Porto. 2021. 90 f. Dissertação (Mestrado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2021.

Novas tendências no cenário de vídeos digitais têm ganhado muita popularidade recentemente, como vídeos de resoluções 4K-8K, vídeos omnidirecionais e conteúdo de tela. Para suportar tais tecnologias, novos codificadores de vídeo têm sido propostos nos últimos anos. Dentre eles, o *Versatile Video Coding* (VVC) é a nova aposta do *Joint Video Experts Team*. Lançado em Julho de 2020, o novo padrão obtém uma eficiência de codificação de 33%, quando comparado ao seu antecessor, isto devido às novas ferramentas implementadas, como estruturas de particionamento flexíveis, novas ferramentas de predição intra blocos, conjunto de novas transformadas, entre outras. Entretanto, tal avanço vem ao custo de um aumento expressivo no tempo do codificação. Outra nova ferramenta implementada é a predição *affine*, que permite a detecção de movimentos não-translacionais durante o processo de predição inter quadros, e portanto, oferece uma eficiência de codificação superior aos padrões anteriores. Entretanto, a predição *affine* acrescenta um aumento no tempo de processamento da predição interquadros de até 47%, dependendo da configuração utilizada, tornando-se um grande desafio para aplicações que necessitam de uma codificação rápida. Sendo assim, este trabalho propõe um esquema para redução do esforço computacional do codificador de vídeo VVC, através do controle de execução da etapa de predição *affine*. Chamado de LEAP (do inglês, *Learning-based Affine Prediction*), o esquema proposto é baseado em aprendizado de máquina, e é capaz de reduzir o tempo total de codificação em 8,49%, e o tempo de codificação do módulo de predição *affine* em 46,94%, em média, com baixas penalidades na eficiência de codificação. Além disso, por se tratar de um controle de uma das etapas de codificação, o esquema proposto pode ser aliado a outras técnicas de redução de esforço computacional para obter uma redução final ainda mais eficiente.

Palavras-chave: codificação de vídeo. aprendizado de máquina. redução de esforço computacional. predição affine.

ABSTRACT

GONÇALVES, Paulo Henrik Ribeiro. **A Learning-based Affine Prediction for VVC Video Coding**. Advisor: Marcelo Schiavon Porto. 2021. 90 f. Dissertation (Masters in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2021.

The new trends in digital video technologies, such as 4K-8K resolution, omnidirectional, and screen content, have become popular in video coding scenarios. To support these technologies, new video encoders have been proposed in recent years. The Versatile Video Coding (VVC) is the most recent video coding standard proposed by the Joint Video Experts Team. Released in July of 2020, the new standard achieves a coding efficiency of 33% higher when compared to its predecessor, the HEVC. This efficiency is achieved by implementing new coding tools, such as flexible partitioning tools, new prediction modes for intra-blocks, and new transform modes. However, the implementation of new tools comes at the cost of a significant encoding time increase. Also, to improve the coding efficiency, VVC implements the affine motion estimation, which allows the detection of non-translational transformations during the interframe prediction. However, the affine motion estimation increases the complexity of VVC interframe prediction up to 47% and becomes an obstacle for real-time applications. In this sense, this work proposes a scheme to reduce the complexity of the VVC encoder by the flow control of the steps of affine motion estimation. Named as LEAP (Learning-based Affine Prediction), the proposed scheme is based on machine learning, and, when implemented in the VVC reference software, it is capable of reducing the total encoding time by 8.49%, and the encoding time of the affine motion estimation by 46.94%, on average, with minor penalties in coding efficiency. Besides, LEAP can be combined with other techniques to improve the complexity reduction of VVC.

Keywords: video coding. machine learning. complexity reduction. affine prediction.

LISTA DE FIGURAS

Figura 1 Estrutura de divisão de um quadro. 20
Figura 2 Redundância de dados espacialmente e temporalmente localiza: (a) quadro atual, (b) quadro vizinho. 21
Figura 3 Codificador de vídeos genérico. 23
Figura 4 Evolução dos principais padrões de codificação ao longo dos anos em relação a redução no *bitrate*. 25
Figura 5 Ilustração da estrutura de particionamento QTMT com representação em forma de árvore e blocos. 29
Figura 6 Exemplo de particionamento do HEVC vs particionamento do VVC. 29
Figura 7 Exemplo de funcionamento da predição interquadros (STORCH, 2020). 31
Figura 8 Fluxograma da ME para uma CU no VVC. 33
Figura 9 Exemplos de modelos *affine* com: (a) 4-parâmetros e (b) 6 parâmetros. 35
Figura 10 Sub-blocos 4×4 em uma CU de tamanho 16×16 codificada com a predição *affine*. 36
Figura 11 Tipos de transformações de bloco possíveis com: (a) 1 PC, (b) 2 PCs e (c) 3 PCs. 37

Figura 12 Tempo médio de execução da ME para cada classe de vídeos. . . . 49
Figura 13 Relação do tempo médio de execução das etapas de predição *affine* para cada classe de vídeos. 51
Figura 14 Porcentagem de execução da etapa de 6-parâmetros em relação à 4-parâmetros (em azul) e porcentagem de vezes em que predição *affine* obteve o melhor MV em toda a ME (em laranja). 52

Figura 15 Fluxograma de execução da ME com o algoritmo LEAP implementado. 55
Figura 16 Sequências de vídeo selecionadas para a etapa de treinamento. . . 57
Figura 17 Valores de *Feature importance* médios obtidos para todos os conjuntos de dados. 61
Figura 18 Comparação dos modelos do LEAP treinadas com algoritmo de Árvores de Decisão e Florestas Aleatórias. 63

Figura 19 Tempo de execução da ME cada sequência de vídeo da classe F. . 81
Figura 20 Tempo de execução da ME cada sequência de vídeo da classe D. . 82
Figura 21 Tempo de execução da ME cada sequência de vídeo da classe C. . 82
Figura 22 Tempo de execução da ME cada sequência de vídeo da classe B. . 83
Figura 23 Tempo de execução da ME cada sequência de vídeo da classe A2. 83

Figura 24 Tempo de execução da ME cada sequência de vídeo da classe A1. 84
Figura 25 Feature Importance de Todos os Parâmetros em Ordem Decrescente. 88

LISTA DE TABELAS

Tabela 1 Lista de sequências de vídeos utilizadas e suas respectivas configurações. 46
Tabela 2 Tempo de execução da ME por etapas. 50
Tabela 3 Resultados obtidos da remoção da etapa de predição affine em relação ao codificador VTM original. 53

Tabela 4 Lista de parâmetros coletados na etapa de mineração de dados. . . 58
Tabela 5 Descrição dos conjuntos de dados após etapa de agrupamento. . . 61
Tabela 6 Resultados de melhores hiperparâmetros e *f1_score* obtidos após aplicação da técnica *Random Search* com 1000 iterações para cada um dos modelos. 64
Tabela 7 Coeficiente de correlação de Person obtido para cada hiperparâmetro. 64
Tabela 8 Resultados de melhores hiperparâmetros e *f1_score* obtidos após aplicação da técnica *Grid Search* para cada um dos modelos. . . . 65

Tabela 9 Resultados obtidos da implementação do algoritmo LEAP proposto em relação ao codificador VTM original. 68
Tabela 10 Estatísticas de predição para cada modelo do LEAP. 70
Tabela 11 Comparação com o trabalho relacionado de Park; Kang (2019). . . 71

Tabela 12 Porcentagem do tempo de execução de cada etapa da ME para todas as sequências de vídeo analisadas. 85

LISTA DE ABREVIATURAS E SIGLAS

ALF	*Adaptive Loop Filter*
AVC	*Advanced Video Coding*
BD	*Bjontegaard Delta*
CABAC	*Context-Adaptive Binary Arithmetic Coding*
CMA	Compensação de Movimento Affine
CTU	*Coding Tree Unit*
CU	*Coding Unit*
DBF	*Deblocking Filter*
FPS	*Frames Per Second*
HD	*High Definition*
HDR	*High Dynamic Range*
HEVC	*High Efficiency Video Coding*
JVET	*Joint Video Experts Team*
LEAP	*Learning-based Affine Prediction*
MC	*Motion Compensation*
ME	*Motion Estimation*
MV	*Motion Vector*
MPEG	*Motion Picture Expert Group*
NETVC	*Internet Video Codec*
OBME	*Overlapped Block Motion Estimation*
PU	*Prediction Unit*
PSNR	*Peak-To-Signal Noise Ratio*
QTMT	*Quadtree with nested Multi-Type*
QP	*Quantization Parameter*
RD	*Rate-Distortion*
RF	*Random Forests*

RTE Redução do Tempo de Execução
SAO *Sample Adaptive Offset*
SBTVD Sistema Brasileiro de Televisão Digital
SD *Standard Definition*
SI *Spatial Information*
TI *Temporal Information*
TU *Transform Unit*
UHD *Ultra High Definition*
UVG *Ultra Video Group*
VCEG *Video Coding Experts Group*
VTM *VVC Test Model*
VVC *Versatile Video Coding*

SUMÁRIO

1 INTRODUÇÃO 14
1.1 Motivação 16
1.2 Objetivos 17
1.3 Organização 17

2 REFERENCIAL TEÓRICO 19
2.1 Conceitos Básicos de Codificação de Vídeo 19
2.2 Codificador de Vídeo 21
2.3 O Padrão *Versatile Video Coding* 24
2.3.1 Configurações de Codificação e Modos de Particionamento de Blocos . 27
2.3.2 A Etapa de Predição Interquadros 30
2.3.3 A Etapa de Predição *Affine* 32
2.4 Aprendizado de Máquina 37
2.5 Trabalhos Relacionados 39
2.5.1 Visão Geral e Comparação Entre Codificadores 39
2.5.2 Algoritmos para Particionamento de Blocos 40
2.5.3 Aprendizado de Máquina na Codificação de Vídeo 41
2.5.4 Algoritmos para predição *affine* 42
2.6 Considerações Finais 43

3 ANÁLISE DO COMPORTAMENTO DA ESTIMAÇÃO DE MOVIMENTO NO PADRÃO VVC 45
3.1 Metodologia de Análise 45
3.2 Métricas Objetivas para Avaliação de Desempenho 47
3.3 Análise do Impacto da Estimação de Movimento no Codificador VVC 48
3.4 Análise das Etapas de Codificação da Estimação de Movimento . . . 49
3.5 Análises da Etapa de Predição *affine* 51

4 LEARNING-BASED AFFINE PREDICTION 55
4.1 Processo Mineração de Dados e Construção dos Conjuntos de Dados 56
4.2 Definição dos Modelos de Aprendizado de Máquina 61
4.3 Treinamento dos Modelos de Florestas Aleatórias 63

5 RESULTADOS EXPERIMENTAIS 67
5.1 Comparação com Trabalhos Relacionados 71

6 CONCLUSÃO 73

REFERÊNCIAS 75

APÊNDICE A TEMPO MÉDIO DE EXECUÇÃO DA ESTIMAÇÃO DE MOVIMENTO PARA CADA SEQUÊNCIA DE VÍDEO 81

APÊNDICE B TEMPO DE EXECUÇÃO DE CADA ETAPA DA ESTIMAÇÃO DE MOVIMENTO 85

APÊNDICE C FEATURE IMPORTANCE DOS PARÂMETROS AVALIADOS . 88

APÊNDICE D TRABALHOS PUBLICADOS DURANTE O PERÍODO DE MESTRADO 89

D.1 Complexity-Aware TZS Algorithm for Mobile Video Encoders 89

D.2 Learning-based Bypass Zone Search Algorithm for Fast Motion Estimation 89

1 INTRODUÇÃO

Com a utilização de dispositivos digitais para visualização, compartilhamento e armazenamento de vídeos, tornou-se necessário o desenvolvimento de técnicas que fornecessem maior eficiência, aliando qualidade e redução do tamanho de dados. Com o advento do computador pessoal em meados de 1980, foram criadas as primeiras técnicas para redução de tamanho de dados de um vídeo bruto para armazenamento nas mídias físicas da época. Tais técnicas tinham como objetivo comprimir o conteúdo de vídeo bruto, podendo ser realizado com ou sem perdas de informação. O conjunto destas técnicas ficou conhecido como codificador de vídeo.

O advento de novas tecnologias de consumo de vídeo digital, bem como as inúmeras exigências dos diversos tipos de conteúdo, impulsionaram o desenvolvimento de novos codificadores de vídeo ao longo dos anos e, por sua vez, a consolidação de padrões[1] de codificação para uso comercial. Com o passar dos anos a busca do público consumidor por novas tecnologias de vídeo, aliado ao interesse da indústria em impulsionar o mercado de entretenimento, levou a popularização de aplicações e serviços voltados para o consumo de vídeo digitais nos mais variados tipos de equipamentos eletrônicos, como, por exemplo, *smartphones*, *tablets*, câmeras digitais, receptores de TV Digital, veículos não tripulados e câmeras microscópicas para soluções médicas.

O *streaming* de vídeo é o serviço mais conhecido e consumido atualmente e, por isso, vem dominando o cenário global de tráfego de dados. Segundo o Relatório Global de Fenômenos da Internet (SANDVINE, 2020), o *streaming* de vídeo lidera o ranking de tráfego de dados na internet com 57,64% do volume total em 2020 (2,20% maior se comparado ao mesmo período no ano anterior). Parte deste crescimento deve-se a adesão das principais aplicações de compartilhamento de vídeo no período da pandemia COVID-19. Com o movimento conhecido como *Fique-Em-Casa*, o Youtube tornou-se a principal fonte de informação dos consumidores, além de apresentar um aumento na procura por conteúdos culinários, *fitness*, *do it yourself*, entre outros. Dados estes fatores, o Youtube tomou a liderança no cenário com 15,94% do tráfego

[1] Na área de codificação de vídeo, um padrão de codificação é um conjunto de ferramentas descritos em um documento formal e aprovado pela comunidade, que descreve o *bitstream* final gerado, permitindo com que ele seja decodificado.

global. O Netflix, que liderava o ranking em 2019, consolidou-se em segundo lugar com 11,42%.

Nas diversas formas de reprodução e compartilhamento de vídeos digitais, nota-se uma diminuição drástica dos conteúdos de definição padrão (SD, do inglês *Standard Definition*) de resoluções 576p (720×576 pixels), e a consolidação dos conteúdos de alta definição (HD, do inglês *High Definition*) de resoluções 720p (1280×720 pixels) e 1080p (1920×1080 pixels), sendo que estes já são encontrados com facilidade na internet, bem como nos televisores e dispositivos móveis. Isto se deve principalmente à facilidade para gravação, manipulação e compartilhamento deste tipo de conteúdo, uma vez que o aumento na capacidade de processamento no dispositivos móveis tornaram estas tarefas viáveis. Entretanto, a constante procura dos consumidores por maior qualidade de vídeo com detalhamento maior faz com que haja também uma necessidade pelo aumento na resolução dos vídeos (SZE; BUDAGAVI; SULLIVAN, 2014). Neste sentido, é comum encontrarmos televisores e smartphones com suporte a vídeos de ultra definição (UHD, do inglês *Ultra High Definition*), como a tecnologia 4K e 8K, que diz respeito às resoluções de 3840×2160 e 7680×4320 pixels, respectivamente. Além disso, as plataformas de *streaming* ofertam cada vez mais seus conteúdos neste tipo de resolução.

Por outro lado, a busca por novas tendências, como vídeos 360º e nuvens de pontos, acarretam em novos desafios no cenário de codificação de vídeo, devido a natureza peculiar deste tipo de conteúdo. Neste sentido, o aumento na resolução dos vídeos digitais torna-se um grande desafio para a área de codificação de vídeo, uma vez que as ferramentas devem ser capazes de suportar este tipo de conteúdo para ofertar uma boa eficiência de compressão. Sem o processo de codificação de um vídeo, armazenar, manipular ou transmitir este tipo de conteúdo tornam-se tarefas completamente inviáveis.

Os codificadores de vídeo atuais são capazes de codificar vídeos de diferentes resoluções e formatos a partir da remoção das redundâncias de dados presentes neste tipo de conteúdo (RICHARDSON, 2002). Entretanto, o aumento no número de ferramentas e a complexidade dos algoritmos de codificação também é um fator importante no desenvolvimento de um codificador. Estes fatores inviabilizam o uso de codificadores de vídeo antigos e, portanto, faz-se necessário a criação de novos codificadores para suportar tais demandas.

Para suportar as demandas citadas anteriormente, um novo padrão de codificação foi proposto pelo *Joint Video Experts Team* (JVET), em 2018. Nomeado de *Versatile Video Coding* (VVC) (CHEN, 2020), o novo padrão traz ferramentas para otimizar a codificação de vídeos naturais, e permitir a codificação e compressão de vídeos 360º e conteúdo de tela. Quando comparado ao seu antecessor, o VVC é capaz de atingir uma eficiência de codificação 33% superior (TAKAMURA, 2019). Entretanto, para tal

feito, é necessário um aumento no tempo de codificação de até 31 vezes (PAKDAMAN et al., 2020), tornando inviável a utilização do VVC na grande maioria de aplicações de vídeo atuais que necessitam de uma codificação em tempo real. Neste sentido, é interessante o estudo de soluções que acelerem o processo de codificação do VVC, sem penalidades significativas na qualidade do vídeo.

1.1 Motivação

A codificação de vídeo é um tópico amplo e abrange diversas áreas da academia e indústria. Optar entre codificadores de vídeo torna-se uma tarefa bastante complexa, uma vez que há diversos codificadores especializados nos mais diversos tipos de tarefas. Entretanto, algo que une todas as implementações é o interesse em oferecer a melhor experiência para o consumidor. Sendo assim, a busca por novas soluções que possam melhorar a eficiência de codificação é um tópico importante, pois impacta diretamente o usuário final, uma vez que a demanda dos consumidores é uma boa relação entre qualidade de vídeo e tamanho do *bitstream*[2]. Entretanto, as tecnologias atuais buscam atender estas demandas, em troca de um alto custo de tempo e esforço computacional.

Neste sentido, reduzir o tempo de codificação é outro tópico importante na área de codificação de vídeo, dado que um codificador deve ser capaz de adaptar-se às diversas demandas da indústria, seja por uma codificação que priorize a qualidade final, ou que sejam capazes de serem codificadas com baixo custo computacional para implementação em dispositivos móveis, ou mesmo em aplicações que exigem codificação em tempo real.

No VVC este cenário é ainda mais desafiador. Dentre as diversas etapas de um codificador, destaca-se a etapa de predição interquadros, que busca remover as redundâncias temporais entre quadros de um vídeo. Dentro da predição interquadros, a etapa com maior destaque é a Estimação de Movimento (ME, do inglês *Motion Estimation*). A ME é a etapa que mais demanda esforço do codificador VVC, podendo ocupar até 57% do tempo de codificação.

Muito deste incremento em esforço computacional deve-se as novas estruturas de particionamentos e as novas ferramentas de predição incluídas no VVC. Uma das ferramentas que se destaca é a predição *affine*, que proporciona uma maior flexibilidade para predição, adaptando-se aos movimentos não translacionais dos objetos em uma cena de vídeo. Ainda segundo Pakdaman et al. (2020), a predição *affine* é responsável por 17% do tempo de codificação do VVC, em média.

[2] *Bitstream* é uma série de bits que armazena as informações necessárias do vídeo original, bem como os sinais de controle de fluxo após o processo de codificação. O *bitstream* é armazenado e/ou enviado a fim de diminuir o armazenamento ou consumo de banda. Uma vez recebido, o *bitstream* não pode ser legível sem passar pelo processo inverso da codificação, também chamado de decodificação.

Existem diversos trabalhos na literatura que estudam o comportamento do VVC e descrevem as ferramentas implementadas. Atualmente, a redução de esforço computacional é um tópico bastante abordado na academia. Por isso, diversos trabalhos com foco na otimização e/ou simplificação das ferramentas de codificação têm sido propostos recentemente. Neste sentido, este trabalho inova apresentando um esquema capaz de otimizar a etapa de ME e reduzir o esforço computacional do codificador VVC.

1.2 Objetivos

Dadas todas as questões apresentadas até o momento, o objetivo primordial desta dissertação é apresentar um esquema para redução do esforço computacional do codificador de vídeo VVC, baseado em aprendizado de máquina. Com foco na predição *affine*, o esquema chamado de *Learning-based Affine Prediction* utiliza-se do algoritmo de Florestas Aleatórias para a definição de modelos que decidem entre: executar a predição *affine* de forma completa, executar a predição *affine* de forma parcial, ou não executar a predição *affine*.

Desta forma, evidenciam-se os objetivos específicos desta dissertação:

1. Apresentar a etapa de estimação de movimento do padrão VVC;
2. Apresentar a ferramenta de predição *affine* na etapa de estimação de movimento;
3. Analisar a complexidade da predição interquadros, com foco na etapa de predição *affine*;
4. Desenvolver um esquema baseado em aprendizado de máquina para redução do esforço computacional empregado na etapa de predição *affine*.

Embora existam trabalhos relacionados que também buscam reduzir o esforço computacional da etapa de ME do codificador de vídeo VVC, este trabalho é o único direcionado a apresentar uma solução exclusiva para a etapa de predição *affine*.

1.3 Organização

Este trabalho está organizado em seis Capítulos. O Capítulo dois apresenta o referencial teórico nos tópicos de codificação de vídeo e aprendizado de máquina necessário para a compreensão e desenvolvimento do trabalho, bem como os trabalhos relacionados nestes tópicos. O Capítulo três apresenta algumas análises realizadas do comportamento da etapa de estimação de movimento e predição *affine*, que sustentam a importância de reduzir o esforço computacional empregado por estas etapas

no codificador VVC. O Capítulo quatro apresenta o esquema para redução do esforço computacional empregado pela etapa de predição *affine*, e descreve os passos necessários para definição, testes e validação dos modelos de Florestas Aleatórias treinados. O Capítulo cinco discute os resultados experimentais coletados a partir das simulações realizadas com o esquema proposto no codificador VVC, bem como uma comparação com trabalho relacionado. Por fim, o Capítulo 6 conclui esta dissertação.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados conceitos de codificação de vídeo que auxiliarão o entendimento do presente trabalho, passando pela definição de um vídeo digital e apresentando um modelo básico de um codificador de vídeo atual. Este capítulo também apresenta o novo padrão de codificação VVC, as principais ferramentas que este codificador implementa, além de apresentar, com maiores detalhes, a etapa do codificador no qual este trabalho é focado, a predição *affine*. Além disso, este capítulo também apresenta os trabalhos na literatura relacionados com a redução de esforço computacional na ME.

2.1 Conceitos Básicos de Codificação de Vídeo

Para podermos tratar dos conceitos para codificação de um vídeo digital é necessário entender primeiramente o que é um vídeo digital. Um vídeo digital pode ser entendido como uma sequência de imagens estáticas e independentes que dão a sensação de movimento do vídeo quando exibidas em uma determinada frequência, ou seja, a uma certa quantidade de imagens por segundo. Essas diferentes imagens que compõem o vídeo são chamadas de quadros. Os quadros são compostos por blocos e estes são compostos por pixels. Um pixel é a unidade básica de um vídeo e é composto por, geralmente, três amostras/canais de informação (RICHARDSON, 2002). Esta estrutura básica pode ser encontrada na maioria dos codificadores de vídeo atuais e é ilustrado na Figura 1.

Um pixel (abreviatura de *Picture Element*, em tradução literal Elemento de Imagem) é uma abstração utilizada para a representação de uma unidade básica de uma imagem (RICHARDSON, 2002). Cada ponto mostrado em uma tela de monitor ou televisor é chamada de pixel, portanto uma imagem é uma matriz de pixels. Dessa forma, uma imagem com resolução de alta definição de 1080×720 pixels contém um total de 777.600 de pixels. Além disso, cada um desses pixels podem assumir apenas uma cor por vez, que é definida pela associação de diferentes canais/amostras.

Normalmente, são utilizados três canais para compor um pixel, como ocorre, por

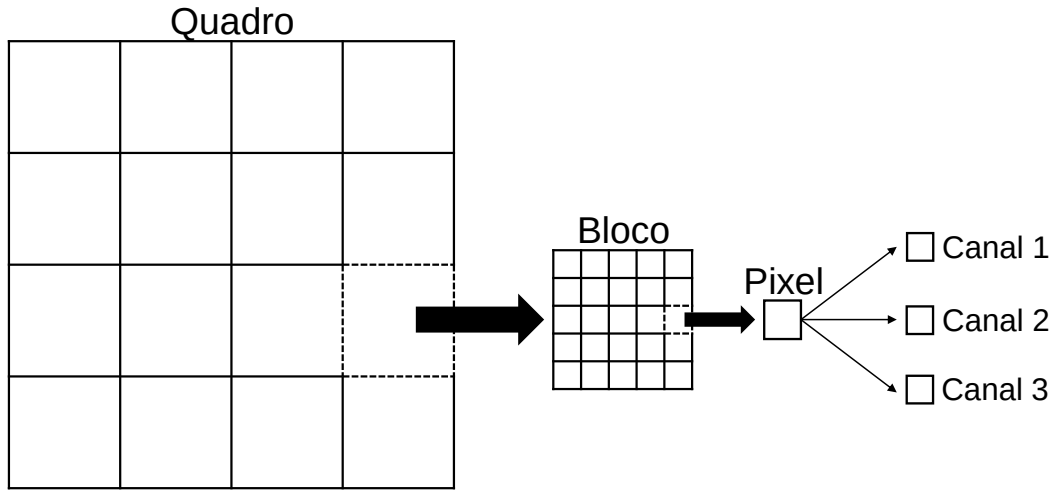


Figura 1 – Estrutura de divisão de um quadro.

exemplo, nos espaços de cores RGB, comumente utilizado para exibição em monitores e televisores. Dessa forma, os canais/amostras ilustrados na Figura 1 como Canal 1, Canal 2 e Canal 3, correspondem a *Red*, *Green* e *Blue* no espaço de cor RGB. O espaço de cor mais utilizado na codificação de vídeo é o YCbCr, onde o Y representa as informações de luminância, Cb as informações de crominância azul, e Cr as informações de crominância vermelha. Assim, cada pixel é formado por uma amostra de luminância e duas amostras de crominância. Este formato é vantajoso pois permite que as informações de luz sejam separadas das informações de cor. Dado que o sistema visual humano é mais sensível às informações de luminância do que crominância, o YCbCr permite que as informações de cor possam ser representadas com um número menor de bits (RICHARDSON, 2010). Por meio desta técnica, uma quantidade significativa de informações de um vídeo pode ser reduzida sem grandes impactos na qualidade visual.

Quadro é a representação de uma imagem estática em um vídeo digital e pode ser entendido como uma matriz de amostras. Neste sentido, o tamanho desta matriz diz respeito à resolução espacial do vídeo. Dessa forma, um vídeo pode ser classificado em uma resolução, que é definida pela sua largura e altura. Dizer que um vídeo possui resolução de 1920×1080 pixels, por exemplo, significa dizer que cada quadro que compõe o vídeo é uma matriz de amostras com essa mesma largura e altura, respectivamente.

Para possibilitar o processo de codificação, um quadro geralmente é dividido em diversos blocos. Sendo assim, pode-se também entender um quadro como sendo um conjunto de blocos. Na codificação de vídeo, um bloco é uma submatriz da grande matriz de amostras que compõem a imagem (RICHARDSON, 2010). Nos padrões de codificação de vídeo atuais, como o VVC, são utilizados mais de um tamanho de bloco, como 128×128 pixels, 64×64 pixels e 32×32 amostras.

Para produzir a sensação de movimento, os quadros devem ser exibidos em uma certa taxa por segundo, mais conhecida por sua sigla em inglês FPS (abreviação para *Frames Per Second*). Sendo assim, a resolução temporal, diferente da resolução espacial, diz respeito à quantidade de quadros exibidos por segundo. Para vídeos de alta definição, como 1920×1080 pixels, a taxa de quadros por segundo, para manter a sensação de movimento, é acima de 24 quadros (ARMSTRONG et al., 2009). Contudo, é comum a utilização de taxas superiores para suavizar ainda mais a sensação de movimento. Nas diversas plataformas de distribuição de vídeos digitais é comum encontrarmos taxas como 60 e 120 quadros por segundo.

2.2 Codificador de Vídeo

Um codificador tem como objetivo reduzir o volume de informação necessária para representar um vídeo digital, a partir da simplificação de informações menos relevantes ao sistema visual humano, e da remoção das redundâncias de dados. Esta redundância diz respeito aos dados que se repetem ao longo de um vídeo. No escopo de vídeos 2D existem três tipos de redundância: redundância espacial, redundância temporal e redundância entrópica (RICHARDSON, 2010). Se o codificador remover as redundâncias, ou seja, as informações repetidas ou similares, somente uma parte das estruturas apresentadas na imagem precisam ser armazenadas, enquanto o restante dos dados que se repetem podem ser removidos. Isso pode ser observado na Figura 2(a), onde partes do quadro apresentam bastante similaridade, como partes da areia, do céu, e até mesmo no mar. Ao fazer a remoção dos dados redundantes, um codificador pode armazenar somente uma referência para a região do próprio quadro onde foi identificada a similaridade. Este processo recebe o nome de codificação intraquadro nos codificadores de vídeos atuais.

Considerando que para termos a sensação de movimento em um vídeo é necessária uma grande quantidade de quadros em um curto intervalo de tempo, muitos dados

(a) Quadro atual (b) Quadro seguinte

Figura 2 – Redundância de dados espacialmente e temporalmente localiza: (a) quadro atual, (b) quadro vizinho.

acabam se repetindo de um quadro para o outro, produzindo assim o que é chamado de redundância temporal. Um exemplo desse tipo de redundância é ilustrado na Figura 2, que apresenta dois quadros vizinhos de uma mesma cena. Como pode ser observado, pouca coisa mudou no quadro atual com relação ao quadro vizinho. Dessa forma, o codificador pode referenciar entre os quadros as regiões que são similares, armazenando dados apenas uma vez e eliminando grande parte da informação redundante. Este processo recebe o nome de codificação interquadros.

No exemplo da Figura 2(b) pode-se notar que todos os blocos no quadro vizinho sofreram um deslocamento suave em relação ao quadro atual. Embora a incidência de luz, ou sombras, possa alterar os valores das amostras, ainda assim é possível identificar que as informações são muito similares ao quadro vizinho. Sendo assim, em vez de representar todas as amostras do quadro atual, pode-se apenas gerar uma referência ao ponto onde a informação similar se encontra no quadro vizinho, e armazenar uma informação diferencial, que diz respeito as mudanças entre as amostras de referência e as atuais.

A redundância entrópica está relacionada com a probabilidade de ocorrência dos símbolos codificados. Desta forma, com uma representação mais eficiente destes símbolos é possível transmitir uma maior quantidade de informação por símbolo. Nesta etapa são utilizados algoritmos de compressão sem perdas.

A qualidade de um vídeo codificado pode ser medida de duas maneiras: qualidade subjetiva e qualidade objetiva. A qualidade subjetiva diz respeito a impressão do espectador sobre o vídeo. Porém, avaliar a eficiência de codificação subjetivamente é uma tarefa bastante difícil. Assim, ao invés da qualidade subjetiva, os codificadores de vídeos atuais utilizam métricas de qualidade objetivas. A métrica mais utilizada é o *Peak-To-Signal Noise Ratio* (PSNR), que é medida em escala logarítmica e expressa em decibéis (dB) (SALOMON, 2007). Além disso, essa métrica é baseada no erro quadrático médio (MSE, do inglês *Mean Square Error*) entre o quadro original e o quadro codificado. As equações 1 e 2 mostram as definições do MSE e do PSNR, respectivamente, onde *R* é uma matriz que representa as amostras do quadro reconstruído, *O* representa as amostras do quadro original, e *h* e *w* diz respeito a altura e largura do quadro, respectivamente. Na equação 2, *MAX* representa o maior valor de uma amostra.

$$MSE(x,y) = \sum_{i=0}^{h-1}\sum_{j=0}^{w-1}(R_{i,j} - O_{i,j})^2 \tag{1}$$

$$PSNR = 10 \times log_{10}\left(\frac{MAX^2}{MSE}\right) \tag{2}$$

Um exemplo de codificador de vídeos genérico é apresentado na Figura 3. Nela pode-se ver que o quadro atual e quadros de referência são entradas do processo de

codificação. O quadro atual é o quadro que está sendo codificado naquele momento, enquanto que o quadro de referência representa um quadro que foi previamente codificado e é utilizado como referência para a codificação do quadro atual. Além disso, a Figura mostra como saída um quadro reconstruído, que é o quadro atual codificado e decodificado para poder ser usado como quadro de referência na codificação dos próximos quadros.

A primeira etapa da codificação, chamada de predição, é composta pela predição intraquadro e pela predição interquadros. Como dito anteriormente, a predição intraquadro é o módulo que visa identificar as redundâncias espaciais presentes em um quadro, enquanto o módulo interquadros é responsável por identificar as redundâncias temporais presentes entre quadros. Estes dois tipos de predição também fazem com que este tipo de codificador seja conhecido como híbrido. É no módulo de predição interquadros que se encontra a etapa de Estimação de Movimento, foco deste trabalho.

Após um bloco do quadro atual ser codificado pela predição intraquadro ou pela predição interquadros, é realizada uma subtração entre os valores do bloco original e o bloco resultante da predição. Essa diferença é chamada de resíduo e é sobre este resíduo que a codificação residual é realizada (RICHARDSON, 2010). Na etapa de codificação residual, primeiramente os resíduos são transformados do domínio espacial para o domínio das frequências e então é realizado um corte/atenuação das frequências menos relevantes ou imperceptíveis ao olho humano, uma vez que estas não influenciam, ou influenciam muito pouco, na experiência do espectador. Estas

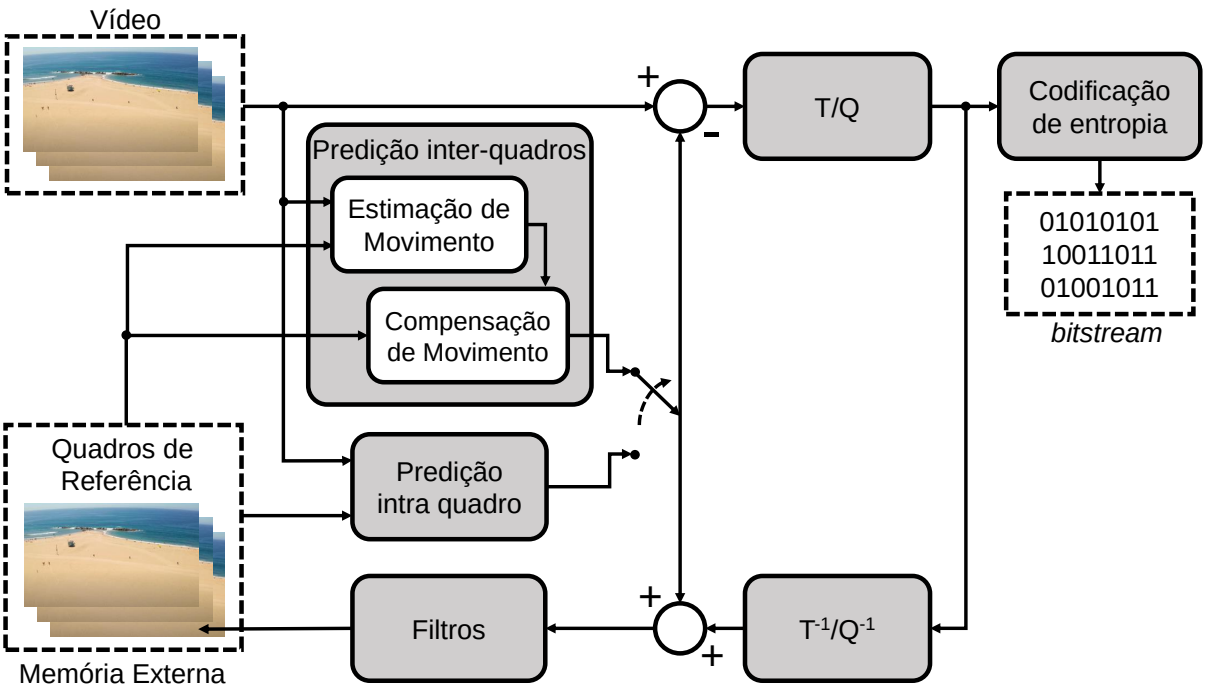


Figura 3 – Codificador de vídeos genérico.

duas etapas são chamadas de transformação e quantização.

Após isso, é realizada a etapa de codificação de entropia. Esta etapa está diretamente relacionada a como os dados são codificados e permite que uma menor quantidade de bits seja utilizada para a representação dos valores com maior probabilidade de ocorrência. São os dados gerados após esta etapa que resultarão no vídeo comprimido, o chamado *bitstream*, que será então transmitido ou armazenado.

É importante destacar que o codificador realiza também uma decodificação a partir dos dados gerados pela etapa de codificação residual. Isto é necessário para a geração dos quadros de referência que serão utilizados na codificação intraquadro do quadro atual e na codificação interquadros de quadros futuros.

2.3 O Padrão *Versatile Video Coding*

Historicamente, a união dos grupos *Motion Picture Expert Group* (ISO/IEC MPEG) e *Video Coding Experts Group* (ITU-T VCEG) obtiveram sucesso na padronização dos codificadores estados-da-arte do mercado. O *Advanced Video Coding* (H.264/AVC) desenvolvido entre os anos de 1997 a 2003 ainda é consolidado tanto no mercado quanto na literatura, e é o atual padrão adotado pelo Sistema Brasileiro de Televisão Digital (SBTVD). Seu sucessor, o *High Efficiency Video Coding* (H.265/HEVC) foi desenvolvido entre os anos de 2005 a 2013 e inseriu diversas novas ferramentas de predição e modos de particionamento de blocos assimétricos. Desde a sua versão inicial, o HEVC implementou diversas extensões (JCT-VC, 2019), fazendo com que o HEVC fosse, por muito tempo, o padrão de codificação estado-da-arte, pois oferecia uma eficiência de codificação maior do que os padrões lançados posteriormente. Com todos estes benefícios, mais de 2 bilhões de dispositivos compatíveis com o HEVC foram distribuídos em todo o mundo.

Impulsionados pelo crescimento dos consumidores nos mais diversos tipos de mídia, tais como o a popularização de conteúdos de mídia 4K e 8K, vídeos *High Dynamic Range* (HDR), vídeos 360º, nuvem de pontos, jogos na nuvem, e conteúdo de tela, recentemente, os grupos MPEG e VCEG uniram-se novamente para criar o mais novo padrão de codificação. Denominados como *Joint Video Experts Team* (JVET), os grupos desenvolveram entre os anos de 2015 à 2020 o *Versatile Video Coding* (H.266/VVC) (CHEN, 2020). O objetivo do JVET com o VVC era ofertar uma eficiência de compressão maior que o seu antecessor, o HEVC, com um bitrate 30-50% menor, mantendo a mesma qualidade subjetiva. O VVC teve sem lançamento em julho de 2020, juntamente com um modelo de teste chamado de VTM (do inglês, VVC *Test Model*) na versão 10.0.

Quando comparado com os demais padrões de codificação do mercado, o VVC consegue superar os concorrentes na eficiência de compressão (TAKAMURA, 2019).

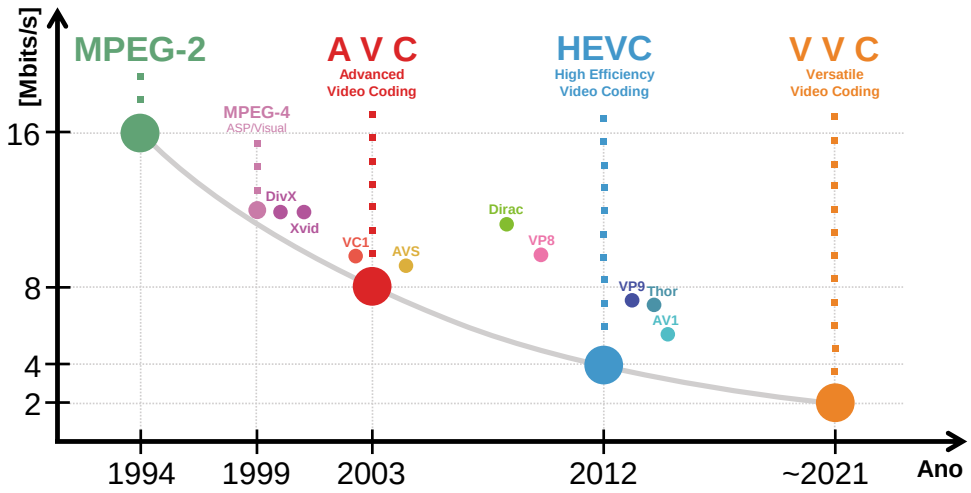


Figura 4 – Evolução dos principais padrões de codificação ao longo dos anos em relação a redução no *bitrate*.

Entretanto este ganho vem ao custo de um acréscimo substancial em esforço computacional. A Figura 4 ilustra a evolução ao longo dos anos dos principais padrões de codificação do mercado, em relação a eficiência de compressão.

Para oferecer maior eficiência de compressão, o VVC inseriu uma série de novas ferramentas, e refinou aquelas já existentes no HEVC. Algumas destas ferramentas são importantes para o escopo desta dissertação e serão discutidos nas próximas seções com maior detalhe. As principais ferramentas de cada etapa do codificador serão resumidas nesta seção:

- **Predição Intraquadro:** Diferentemente do HEVC que permite somente 35 modos de predição intra, o VVC implementa 67 modos, sendo 65 modos direcionais, um modo DC e um modo planar. Este incremento em relação ao HEVC permite uma maior angulação para refinamento da predição. Além disso, o VVC implementa três novas maneiras de realizar a predição intraquadro de um bloco: (1) A *Cross-Component Linear Model*, na qual as amostras de Chroma são preditas baseadas nas amostras de Luma reconstruídas para a mesma CU, usando um modelo linear; (2) O *Intra Sub-Partitions*, onde as amostras de Luma são divididas em duas ou quatro subpartições, tanto horizontalmente quanto verticalmente. Todas as subpartições compartilham o mesmo modo intra, no entanto, o processamento é realizado gradualmente, de cima para baixo ou da esquerda para a direita, de modo que cada uma utilize as amostras reconstruídas anteriormente para gerar a predição da subpartição atual; (3) A *Matrix-based Intra Prediction* seleciona uma linha de amostras vizinhas reconstruídas, a partir dos blocos esquerda-cima, como vetores de entrada, e após um pré-processamento, executa a interpolação linear tanto na direção horizontal quanto vertical. Por fim, diferentemente do HEVC, o VVC permite que a predição intra seja aplicada a

blocos retangulares e com resoluções maiores. Além disso, o VVC estende as amostras de referência habilitando o uso de diversas linhas de referência, com a finalidade de melhorar a qualidade da predição.

- **Predição Interquadros:** Nesta etapa, o modo de predição bilateral é estendido para comportar até cinco pesos pré definidos, além da média ponderada simples implementada no HEVC, esta técnica é chamada de *Generalized Bi-prediction*. Além disso, o *BiDirectional Optical Flow* pode aplicar um refinamento a nível de pixel em cima desta predição, para obter resultados ainda melhores. Outro melhoria é o processo de estimação de movimento fracionária, que pode ser aplicada em uma predição de 1/16 nas amostras de Luma. Além disso, o VVC introduz um novo método de predição triangular, capaz de dividir uma CU em outras duas triangulares, tanto na diagonal quanto na diagonal inversa. Por fim, um modo de predição de movimentos *affine* é implementada para lidar com os movimentos irregulares, ou não translacionais, presentes no vídeo. Esta ferramenta é importante para a elaboração do trabalho, e será discutida com mais detalhes na seção 2.3.3.

- **Transformada:** O VVC aplica a etapa de transformada em blocos de maior resolução, de , e, para explorar ainda mais a redundância espacial, o VVC introduz uma transformada secundária com resolução menor.

- **Codificação de Entropia:** Para uma codificação de entropia mais eficiente, o VVC utiliza uma versão otimizada do *Context-Adaptive Binary Arithmetic Coding* (CABAC). Desta forma, a seleção de modelos de probabilidade para elementos de sintaxe depende do valor e do número de elementos diferentes de zero em uma vizinhança local.

- **Filtros *In-Loop*:** Para oferecer uma qualidade subjetiva ainda melhor, o VVC implementa três filtros *In-Loop*: *Deblocking Filter* (DBF), *Sample Adaptive Offset* (SAO) e *Adaptive Loop Filter* (ALF). Enquanto o DBF e o SAO são muitos similares às implementações no HEVC, o ALF apresenta-se como a grande novidade, e é usado somente no VVC. O ALF usa um esquema de classificação de blocos para escolher entre 25 filtros diferentes, com base na direção e no nível dos gradientes locais. Esse filtro é aplicado após o DBF e SAO, e somente em blocos com resolução mínima.

As novidades com relação as novas estruturas de particionamento de blocos e as novas ferramentas implementadas na predição interquadros são relevantes para o escopo deste trabalho e, portanto, são discutidos nas seções a seguir.

2.3.1 Configurações de Codificação e Modos de Particionamento de Blocos

Assim como o HEVC, o VVC é composto por três configurações de codificação, cada uma desenvolvida para atingir um conjunto de requisitos. Estas configurações são *All Intra*, *Low Delay* e *Random Access*. Estas configurações são escolhidas antes do início do processo de codificação e são mantidas até o fim. Na configuração *All Intra*, todos os quadros do vídeo são codificados utilizando exclusivamente a predição intraquadro, ou seja, não existem dependências de dados entre quadros codificados. Nos padrões anteriores, por possuir somente a predição intraquadro, a configuração *All Intra* apresentava uma baixa taxa de compressão e, consequentemente, uma relação menor de complexidade quando comparado à predição interquadros. Entretanto, no VVC tem-se um cenário inusitado, uma vez que o modo *All Intra* apresenta uma maior complexidade do que os demais modos. Segundo Pakdaman et al. (2020), a configuração *All Intra* apresenta em média um aumento na complexidade de 1,3$\times$ e 1,4$\times$, quando comparado as configurações *Low Delay* e *Random Access*, respectivamente. Muito deste incremento deve-se principalmente ao número maior de modos de predição intra, assim como descrito na seção 2.3.

Quando utilizado a configuração *Low Delay*, o primeiro quadro do vídeo é codificado com a predição intraquadro, enquanto que os quadros seguintes são codificados com a predição interquadros, fazendo com que o quadro do meio possua dependências em relação a todos os quadros anteriores. Desta forma, nesta configuração os quadros são codificados na mesma ordem que serão exibidos ao usuário, e por isso, apresenta uma boa taxa de compressão. Por estas características, esta configuração é utilizada em aplicações que demandam codificação em tempo real.

A configuração *Random Access* é similar a codificação *Low Delay*, no que diz respeito à forma com que os quadros de referência são dispostos. Quando a configuração *Random Access* é utilizada, o codificador emprega alguns pontos de ancoragem durante a codificação através de um contador de intervalo, também chamado de *IntraPeriod*. O *IntraPeriod* determina quantos quadros serão codificados com a predição interquadros até que um quadro seja codificado exclusivamente com a predição intraquadro. Esta técnica permite que quadros intermediários possuam dependências com apenas alguns poucos quadros, aumentando principalmente a robustez para transmissão de vídeo. Além disso, a configuração *Random Access* não codifica os quadros na mesma ordem que serão exibidos ao usuário. Ao invés disso, os quadros são codificados em uma ordem de vai-e-vem, dentro de um *IntraPeriod*. Devido a estas características, esta técnica melhora o resultado da predição em relação à oclusões e descobrimentos de objetos em cena, melhorando a eficiência de codificação.

A principal inovação do padrão HEVC em relação aos padrões anteriores se dá na sua estrutura de particionamento flexível. Neste sentido, o VVC estende essa estrutura, adicionando ainda mais possibilidades de particionamentos que permitem

uma melhor representação dos objetos em cena.

Atualmente, o VVC suporta um tamanho de bloco maior que o HEVC, onde cada quadro de entrada pode ser particionado em blocos de tamanho máximo de 128×128. Cada um desses blocos é chamado de *Coding Tree Unit* (CTU), e pode ser particionado recursivamente em blocos de tamanhos menores, chamados de *Coding Units* (CUs). Essa estrutura de particionamento é chamada de *Quadtree with nested Multi-Type* (QTMT).

Na estrutura de particionamento QTMT, a CTU atual é primeiramente particionada em quatro CUs quadradas de tamanho iguais, seguindo a estrutura de particionamento de árvore quadrática (QT). Após esse processo, cada CU pode ser recursivamente particionada tanto em outras CUs quadradas usando a QT, quanto em CUs retangulares usando a estrutura de *Multi-Type Tree* (MTT). A MTT permite que cada CU possa ser divida em até quatro tipos de árvores: horizontal binária (BTH), vertical binária (BTV), horizontal ternária (TTH), e vertical ternária (TTV). A divisão de árvore binária (BT, do inglês *Binary Tree*) divide a CU atual em outras duas de forma simétrica, enquanto que a árvore ternária (TT, to inglês *Ternary Tree*) divide a CU atual em outras três, sendo que a primeira, a segunda e a terceira CU possuem 25%, 50% e 25% do tamanho da CU atual, respectivamente. Tanto as estruturas de divisão BT quanto TT podem ser aplicadas tanto na direção horizontal quanto na vertical. Entretanto, para limitar a complexidade deste processo, o VVC delimita que a estrutura de particionamento MTT seja aplicada somente nos nodos folhas da estrutura de particionamento QT. Isto significa que após a MTT ser usada em uma CU, não é permitido nenhum outro particionamento da QT (CHEN, 2020).

A Figura 5 ilustra a estrutura QTMT para uma CTU de tamanho 128×128 dividida em diversas CUs de diferentes níveis de QT e MTT. A linha preta sólida representa os particionamentos da QT, enquanto as linhas pontilhadas representam os particionamentos da MTT (azul para o particionamento binário e laranja para o particionamento ternário). Os nós folha da QT e MTT indicam as CUs e também são usadas para as etapas de predição e transformadas. Dessa forma, diferentemente do HEVC, na estrutura QTMT não há separação de conceitos de Predictions Unit (PU) e Transform Unit (TU).

Com a estrutura de particionamento QTMT, o VVC permite uma maior flexibilidade dos tipos de particionamento de blocos para adaptar as ferramentas para os mais diversos tipos de padrões de textura, e com isso, ofertar uma melhor eficiência de compressão. A Figura 6 mostra na esquerda um exemplo com a estrutura de particionamento encontrada no HEVC, e a direita o mesmo exemplo codificado com a QTMT. É possível perceber a maior flexibilidade para representação das texturas principalmente na região de bordas e nas regiões com maior detalhamento.

Considerando desde o tamanho de bloco máximo de 128×128 até o tamanho mí-

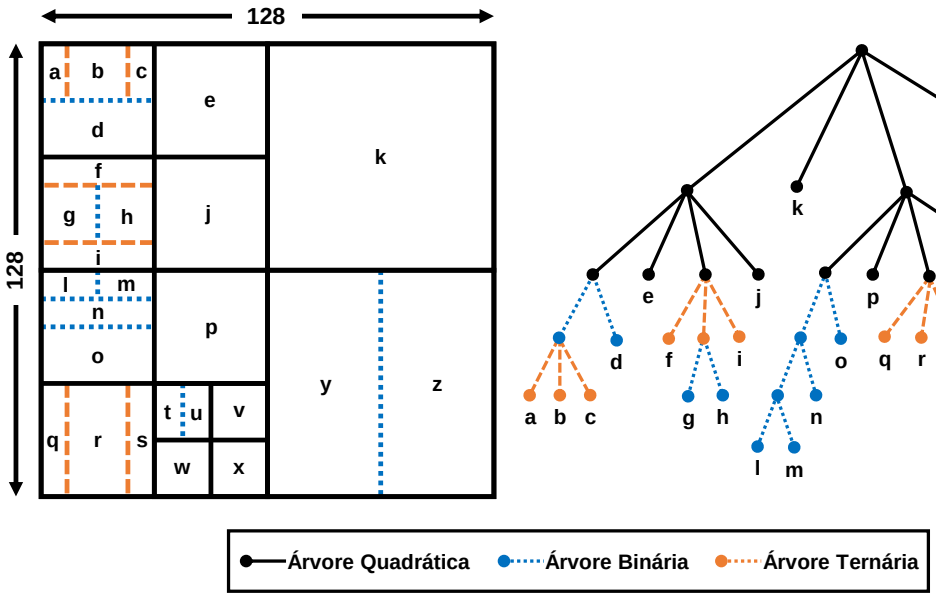


Figura 5 – Ilustração da estrutura de particionamento QTMT com representação em forma de árvore e blocos.

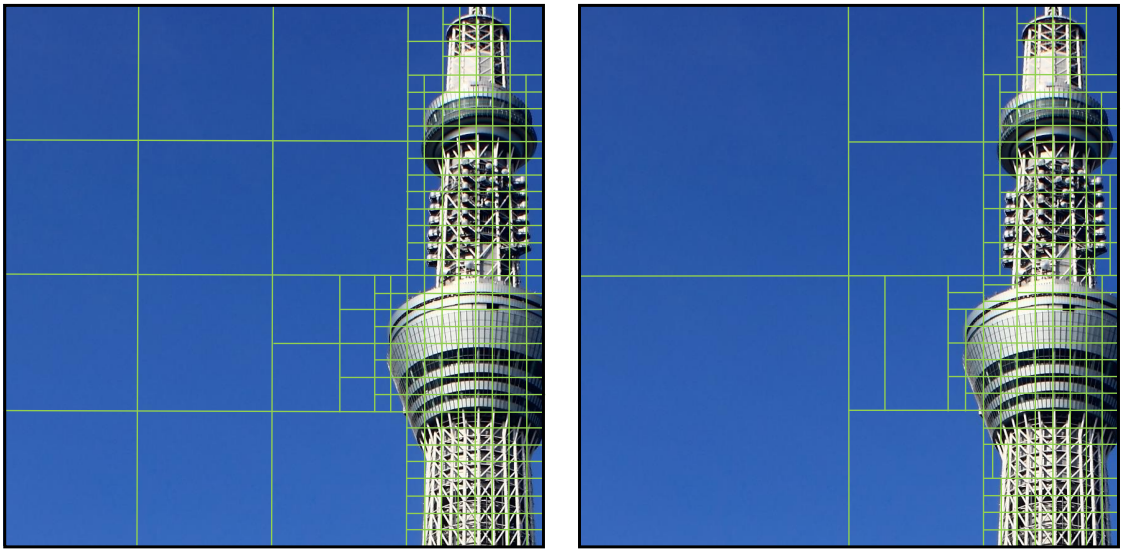


Figura 6 – Exemplo de particionamento do HEVC vs particionamento do VVC.

nimo permitido de 4×4 (CHEN, 2020), pode-se dizer que a QTMT é aplicada em até 6 níveis de particionamento. Entretanto, tal flexibilidade insere uma grande complexidade no codificador VVC, uma vez que é necessário realizar todo o processo de predição para cada possível nó folha. Neste sentido, Pakdaman et al. (2020) realiza um experimento limitando a profundidade da QTMT em um até quatro níveis, em comparação com os seis níveis definidos como padrão. Resultados experimentais apresentaram uma redução do esforço computacional de 82%, 76%, 58% e 26%, respectivamente. Segundo o autor, todos os casos apresentaram efeito indesejável na eficiência de codificação, porém, os experimentos mostraram que técnicas para podar

a QTMT de maneira mais inteligente podem oferecer uma grande redução no esforço computacional do VVC.

2.3.2 A Etapa de Predição Interquadros

Como citado anteriormente, a etapa de predição interquadros consiste em identificar as redundâncias temporais encontradas entre quadros. A etapa de predição interquadros do VVC é muito similar a predição implementada no HEVC. Igualmente, esta etapa necessita do uso de quadros de referências já codificados anteriormente para realizar os algoritmos de busca.

O conjunto de quadros de referência está relacionado a configuração de codificação escolhida. Caso a configuração *Low Delay* esteja sendo utilizada, o conjunto de quadros de referência será composto exclusivamente por quadros temporalmente anteriores ao quadro sendo codificado no momento. Porém, caso a configuração *Random Access* esteja sendo utilizada, o conjunto de quadros de referência pode ser composto por quadros que estão temporalmente tanto à frente quanto atrás do quadro sendo codificado. Os quadros de referência são armazenados em dois *buffers*, chamados de lista L_0 e L_1. A lista L_0 contém os quadros de referência temporalmente passados, enquanto a lista L_1 contém os quadros de referência temporalmente futuros. Cada lista pode conter até 16 quadros de referência, porém, mais de uma instância do mesmo quadro pode ser armazenados nas listas. Isto significa que para maximizar as listas de referência o codificador deve adicionar o mesmo quadro de referência mais de uma vez. O codificador pode escolher fazer isso para poder realizar a predição com pesos diferentes. Esta técnica é chamada de *weighted prediction*. Desta forma, o número máximo de quadros exclusivos nas listas de referências é 8.

A predição interquadros pode ser descrita em duas principais etapas, a Estimação de Movimento (ME, do inglês *Motion Estimation*), e a Compensação de Movimento (MC, do inglês *Motion Compensation*). Na ME, o codificador utiliza-se de algoritmos de busca para encontrar a melhor correspondência, ou similaridade, de blocos nos quadros de referência. Uma vez encontrado, o bloco com maior similaridade, ou bloco predito, é referenciado a partir do deslocamento translacional no quadro de referência. Este deslocamento é mapeado por um vetor de referência, chamado de vetor de movimento (MV, do inglês *Motion Vector*), que caracteriza o deslocamento horizontal e vertical do bloco predito em relação ao bloco atual. Por motivos de desempenho, a busca pelo bloco de referência é realizada dentro de uma área delimitada em cada quadro de referência, chamada de área de busca. Na MC, os MVs e os dados da decisão dos modos de predição obtidos na ME, são aplicados para reconstruir o bloco predito e gerar as informações residuais que serão transmitidos para as demais etapas da codificação (CHEN, 2020).

A Figura 7 apresenta um diagrama da predição interquadros. Nesse caso, o bloco

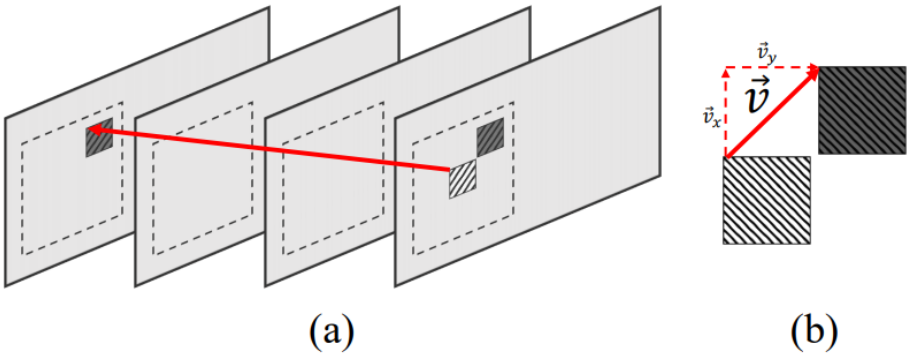


Figura 7 – Exemplo de funcionamento da predição interquadros (STORCH, 2020).

.

hachurado claro representa o bloco a ser predito no quadro atual (o mais a direita na Figura 7(a), enquanto que o bloco hachurado escuro representa o bloco predito após o processamento dos algoritmos de busca aplicados na área de busca, delimitada pelas linhas tracejadas. O bloco predito é referenciado através de um MV $\vec{v}$, caracterizado pela linha vermelha. O MV $\vec{v}$ possui duas componentes v_x e v_y, que representam o deslocamento horizontal e vertical, respectivamente, do bloco a ser predito em relação ao bloco de referência. A decomposição de $\vec{v}$ em v_x e v_y é apresentada na Figura 7(b). Por fim, o MV $\vec{v}$ é utilizado na etapa de MC para reconstrução do bloco predito e geração das informações residuais, que serão transmitidos para as próximas etapas de codificação.

Para determinar a similaridade entre blocos, o VVC busca nos quadros de referência por blocos que minimizem o custo em taxa de distorção (RD, do inglês *rate-distortion*). Este método busca por otimizar a quantidade de distorção (perda da qualidade do vídeo) em relação à quantidade de dados (taxa) necessário para codificar o vídeo (HOANG; LONG; VITTER, 1998). Como mostra a Equação 3, o método de multiplicadores de Lagrange é usado para calcular o custo RD *J*, onde *D* representa o custo da distorção do bloco, λ representa o multiplicador Lagrange, e *R* representa o número estimado de bits para representar o bloco.

$$J = D + \lambda \times R \tag{3}$$

A ME é a etapa que mais demanda esforço computacional nos codificadores de vídeo atuais. No VVC a ME pode ser responsável por até 57% do tempo de codificação, dependendo da configuração escolhida (PAKDAMAN et al., 2020). Isto deve-se principalmente aos algoritmos de busca que já eram implementados no HEVC, mas que foram refinados para o VVC, bem como à inclusão de novas ferramentas de predição.

Neste sentido, a ME no codificador de vídeo VVC pode ser dividida em três etapas principais: A predição unilateral, a predição bilateral e a predição *affine*.

- **Predição unilateral:** Nesta etapa é avaliado cada quadro de referência, a fim de definir o centro da área de busca e executar os algoritmos de buscas para encontrar o bloco com maior similaridade. Este bloco também é chamado de bloco candidato. Um vez avaliado todos os quadros de referência, os candidatos com menores custos RD da lista L_0 quanto da L_1 são selecionados.
- **Predição bilateral:** Melhora o bloco candidato com base nos melhores MVs encontrados na etapa anterior. Primeiramente, o melhor bloco da lista L_0 é selecionado. Em seguida, é realizado uma soma ponderada com o melhor candidato da lista L_1 a fim de gerar um bloco com custo RD menor aos candidatos da etapa unilateral. A predição bilateral é eficaz para sequências de vídeo em que há rápida movimentação, câmeras panorâmicas, *zooming* e mudanças de cena.
- **Predição *affine***: Única novidade do VVC na etapa de ME em relação ao HEVC. Diferentemente das demais etapas, a predição *affine* busca mapear os movimentos não-translacionais de objetos nas cenas, como rotação, cisalhamento e *zooming*. A predição *affine* é foco deste trabalho e será melhor discutida na próxima seção.

A Figura 8 mostra um fluxograma de execução ME para uma CU. Ao final da execução da ME, a etapa com menor custo RD é sinalizada como modo de predição escolhido, e as informações necessárias são enviadas para as etapas posteriores da codificação.

2.3.3 A Etapa de Predição *Affine*

Na geometria euclidiana, uma transformação afim (*affine transformation*) é um tipo de transformação geométrica que preserva linhas e paralelismos, mas não necessariamente distâncias e ângulos (BRANNAN; ESPLEN; GRAY, 1999). Mapear este tipo de transformação no cenário de codificação de vídeos 2D é um desafio que há décadas vêm sido debatido na academia. Neste sentido, o VVC inovou ao implementar um módulo dedicado a este tipo de operação, dado que transformações *affine* são inerentes ao conteúdo de vídeos digitais.

O módulo de predição *affine* é a única novidade do VVC na etapa de ME em relação ao HEVC. Entretanto, este tópico já vem sido debatido na academia desde o início das pesquisas em codificação de vídeo. As primeiras discussões sobre este tema vieram em (AGUI; OKAWA; NAKAJIMA, 1989) e (SULLIVAN; BAKER, 1991), onde os autores descobriram que a compensação de movimento translacional não pode lidar com movimentos mais complexos, como *zooming* ou rotação, de forma eficiente

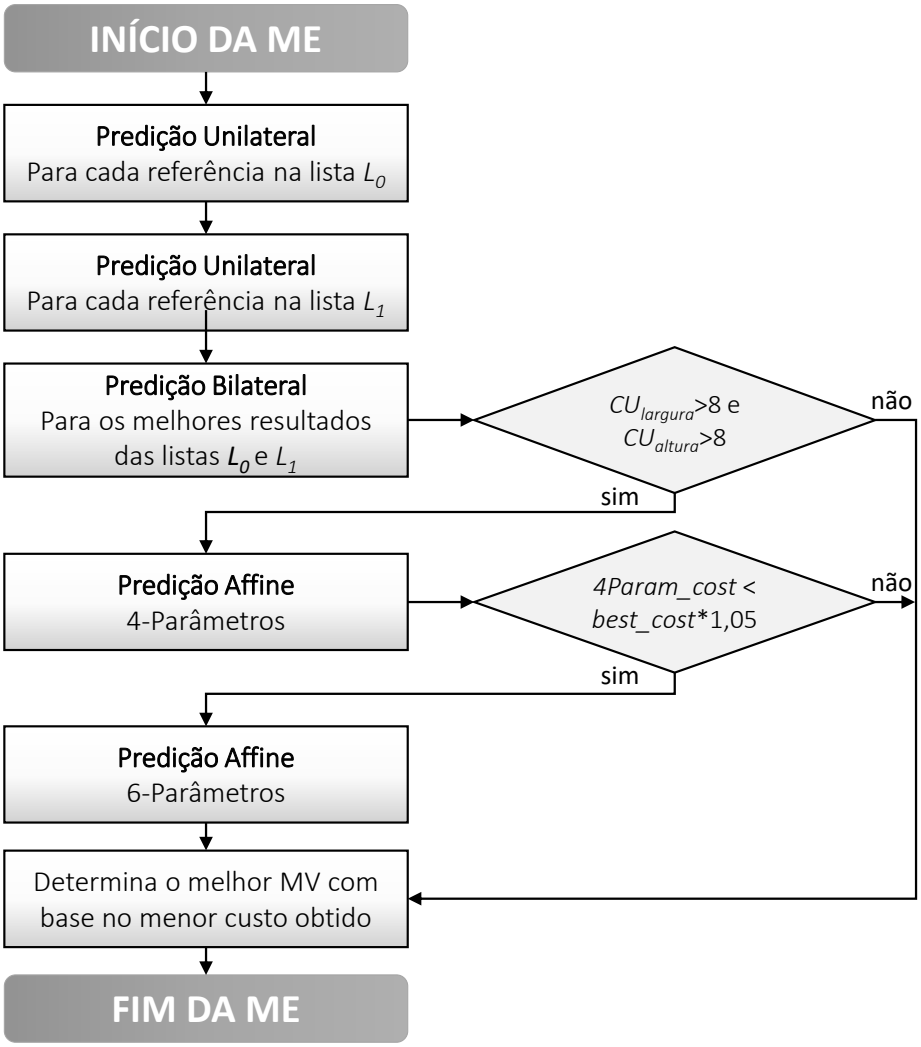


Figura 8 – Fluxograma da ME para uma CU no VVC.

e, portanto, eram necessários modelos mais sofisticados para lidar com movimentos não translacionais. Mais tarde, Seferidis; Ghanbari (1993) e Lee; Wang (1995) propuseram algoritmos de casamento de blocos deformáveis, também chamados de generalizados, nos quais um MV em qualquer posição dentro de um bloco pode ser calculado por um modelo polinomial envolvendo vários MVs em pontos de controle. Estes algoritmos são tão gerais que podem representar uma extensa gama de modelos de movimento complexo. Entre esses modelos, os pesquisadores gradualmente perceberam que o modelo *affine* pode caracterizar a maioria dos movimentos não-translacionais em sequências de vídeo naturais. Desde então, a compensação de movimento *affine* (CMA) atraiu a atenção por quase três décadas.

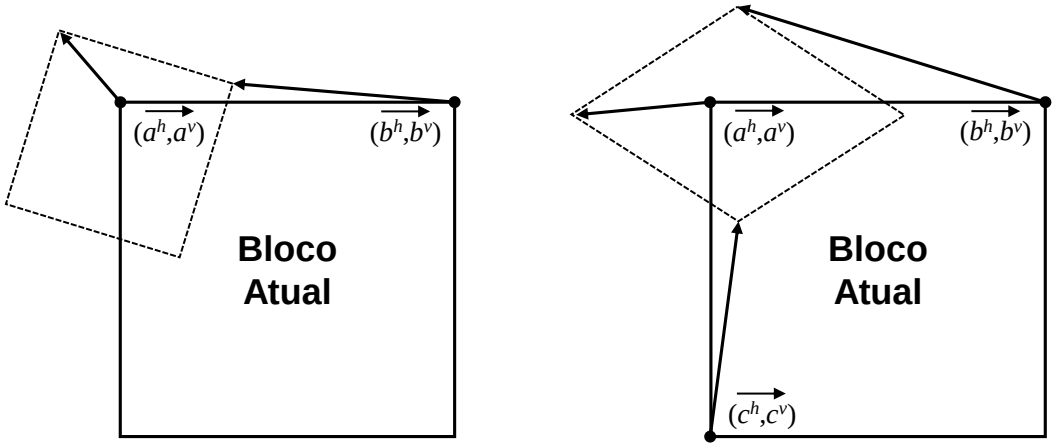
Os autores em (WIEGAND; STEINBACH; GIROD, 2005) e (LI et al., 2005) propõem maneiras diretas de introduzir CMA em codificadores de vídeo existentes e, por isso, também são chamados de CMA global. Para isto, eles assumem que todos os movimentos de uma imagem seguem um modelo *affine* consistente. Com este modelo,

várias imagens de referência sintéticas podem ser construídas a partir de candidatos *affine*. Estes métodos funcionam bem quando a imagem como um todo está girando ou ampliando, mas dificilmente podem lidar com casos onde algumas regiões da imagem estão em conformidade enquanto outras possuem movimentos *affine*.

Os autores em (NAKAYA; HARASHIMA, 1994), (HUANG; HSU, 1994) e (ALTUNBASAK; TEKALP, 1997) propõem esquemas em CMA local. Este tipo de implementação fornece mais flexibilidade, permitindo que movimentos em diferentes regiões sejam derivados por diferentes modelos *affine*. Nestes casos, a imagem é dividida por "malhas", que segmentam regiões com diferentes modelos *affine*. MVs nos vértices, também conhecidos como pontos de controle (PCs), de uma região são usados para derivar todos os MVs de dentro da mesma região. Uma vez que os PCs são compartilhados por regiões vizinhas, é bastante difícil descobrir os MVs nos PCs devido à dependência não casual.

Como conduzir a ME em um esquema CMA tem sido um problema desafiador, uma vez que os métodos de ME tradicional empregam uma complexidade extremamente alta para buscar por MVs nos modelos *affine*. Por isso, embora o CMA seja um tópico de pesquisa popular há muito tempo, ele não foi considerado uma ferramenta de codificação viável nos padrões de codificação de vídeo até nos anos recentes. Existem duas barreiras principais que dificultam as aplicações CMA. Em primeiro lugar, a maioria das soluções CMA impõem uma enorme carga de esforço computacional tanto no codificador quanto no decodificador, o que é uma desvantagem enorme nas aplicações que necessitam de performance. Em segundo lugar, não é fácil projetar uma estrutura eficiente e elegante que possa incorporar o CMA em um sistema padrão de codificação híbrida orientado ao modelo de particionamento em bloco.

Esta situação começou a mudar recentemente, após a finalização do HEVC, quando os pesquisadores começaram a perceber que a eficiência de codificação que é possível ganhar com o CMA se torna mais atrativa na medida que os métodos tradicionais já não são mais tão fáceis de serem melhorados. Além disso, novos estudos no HEVC mostraram que as duas barreiras citadas anteriormente podem ser superadas com as tecnologias atuais. Huang et al. (2013) propôs um modelo *affine* de seis parâmetros, representados por MVs em três pontos distintos. Embora o modelo em si não seja novo, este trabalho abre uma porta para incorporar os PCs na representação de MVs do HEVC. Zhang et al. (2017) incorpora a CMA como um caso de *merge* especial no HEVC. Por fim, Li et al. (2018) simplifica o método de representação de seis parâmetros, e propõe um método de quatro parâmetros. Os modos de predição *affine* e *merge affine* são projetados como algoritmos de predição. Além disso, a granularidade pode ser ampliada do nível de pixel para nível de sub-bloco 4×4, para redução da complexidade desta operação. Como o *tradeoff* entre a complexidade e o desempenho da codificação é atraente, o JVET decidiu por implementar os métodos de CMA



(a) (b)

Figura 9 – Exemplos de modelos *affine* com: (a) 4-parâmetros e (b) 6 parâmetros.

por PCs no projeto do VVC em julho de 2018 (LIN et al., 2018). Após décadas de grandes esforços, o CMA finalmente pode ser incluído em um padrão de codificação de vídeo de forma prática.

No codificador VVC, os modelos *affine* podem ser gerados tanto na predição 4-parâmetros quanto 6-parâmetros. A Figura 9 ilustra as duas formas de predição dos modelos *affine*.

Desta forma, dada qualquer amostra (*x*,*y*) em um bloco a ser predito, é possível definir um MV com coordenadas (mv^h,mv^v), onde mv^h e mv^v representam a posição da amostra predita no eixo-x e eixo-y, respectivamente. Com 4-parâmetros (Figura 9(a)), dois MVs localizados nos cantos esquerdo-cima (a^h,a^v) e direita-cima (b^h,b^v) são definidos como PCs e, a partir destes, qualquer amostra (*x*,*y*) dentro do bloco a ser predito pode ser resolvida pela Equação 4:

$$\begin{cases} mv^h = x\frac{b^h-a^h}{w} + y\frac{b^v-a^v}{w} + a^h \\ mv^v = x\frac{b^v-a^v}{w} + y\frac{b^h-a^h}{w} + a^v \end{cases} \tag{4}$$

onde *w* representa a largura do bloco a ser predito. Com a adição de mais um MV localizado no canto esquerda-baixo (c^h,c^v), é possível definir o modelo 6-parâmetros (Figura 9(b)), onde qualquer amostra (*x*,*y*) dentro do bloco a ser predito pode ser resolvida pela Equação 5:

$$\begin{cases} mv^h = x\frac{b^h-a^h}{w} + y\frac{c^h-a^h}{h} + a^h \\ mv^v = x\frac{b^v-a^v}{w} + y\frac{c^v-a^v}{h} + a^v \end{cases} \tag{5}$$

onde *h* representa a altura do bloco a ser predito.

Embora, em teoria, cada amostra dentro do bloco a ser predito possa derivar seu próprio MV seguindo as Equações 4 ou 5, na prática, realizar a tarefa de MC para

cada pixel torna-se uma tarefa extremamente complexa e inviável. Para evitar este cenário, um nível de sub-bloco é utilizado no VVC. Conforme mostrado na Figura 10, o MV no centro de cada sub-bloco 4×4 é derivado a partir do mv^h e mv^v, e o resultado é compartilhado por todas as amostras do sub-bloco. Com os MVs de todos os sub-blocos, uma técnica chamada de *Overlapped Block Motion Estimation* (OBMC) é aplicada para suavizar os artefatos de blocagem causados pelos sub-blocos (CHEN, 2020). Ainda, para otimizar a performance da predição *affine*, o VVC permite que somente CUs iguais ou maiores que 16×16 realizem essa etapa, conforme mostrado na Figura 8.

Tanto os modelos de predição 4-parâmetros quanto 6-parâmetros podem definir diversas transformações *affine*, bem como composições entre elas. A Figura 11 ilustra os principais tipos de transformações *affine* possíveis utilizando ambos modelos.

Para determinar os PCs utilizados em cada modelo *affine*, o VVC implementa dois novos modos, chamados de modo inter e modo *merge*. No modo inter, os MVs de cada PC de uma CU são apenas sinalizados do codificador para o decodificador, e um algoritmo é projetado para encontrar os MVs preditos em cada PC. Além disso, uma estratégia de ME baseada em gradiente é utilizada no codificador para estimar os PCs (CHEN, 2020). Com o modo *merge*, os MVs de cada PC são derivados de CUs vizinhas. Este trabalho tem como foco principal o modo inter da predição *affine*, pois é empregada dentro da etapa de ME.

No fluxograma de execução da ME para o codificador VVC mostrado na Figura 8, a predição *affine* sempre realiza inicialmente a etapa de 4-parâmetros, desde que este modo esteja habilitado nas configurações de entrada e as condições de tamanho de CU sejam atendidas. Caso seja verificado que o custo obtido após a execução da etapa de 4-parâmetros da predição *affine* (*4Param_cost*) é até 5% maior que o menor custo obtido nas etapas anteriores de predição unilateral e bilateral (*best_cost*), então

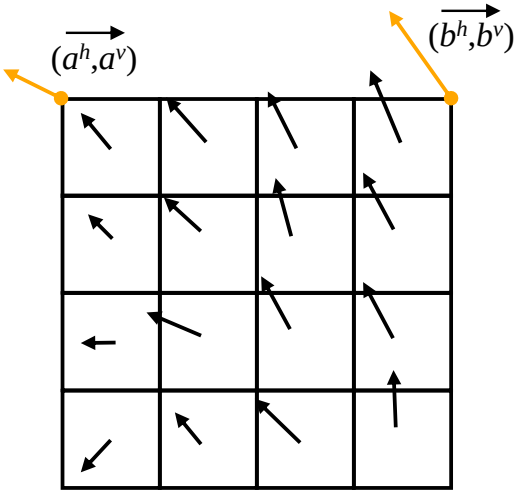


Figura 10 – Sub-blocos 4×4 em uma CU de tamanho 16×16 codificada com a predição *affine*.

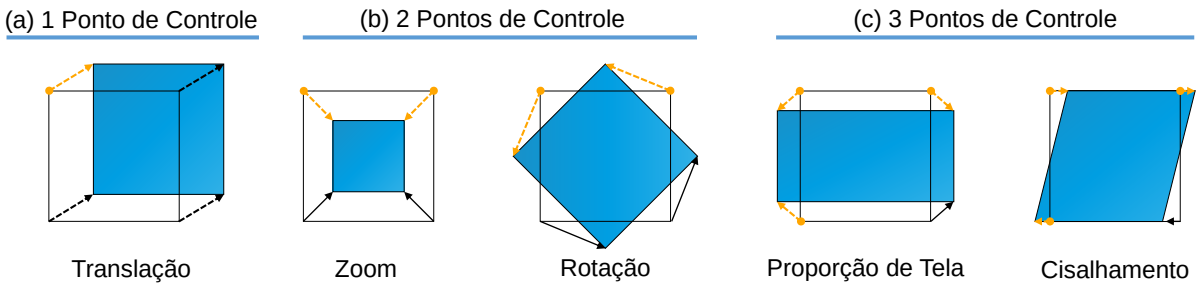


Figura 11 – Tipos de transformações de bloco possíveis com: (a) 1 PC, (b) 2 PCs e (c) 3 PCs.

a etapa de 6-parâmetros é executada. Esta é uma maneira que o codificador adota para identificar os casos onde a predição *affine* tem potencial de obter bons resultados e, ao mesmo tempo, restringir a complexidade empregada na execução destas etapas.

Dadas todas as informações nesta seção, é possível observar que mesmo com a evolução dos codificadores de vídeo nos últimos anos, realizar etapas de predição *affine* na ME ainda é muito custosa para um codificador. Neste sentido, este trabalho busca avaliar as condições da etapa de predição *affine* dentro da etapa de predição interquadros do VVC, e propor uma solução capaz de reduzir a carga de complexidade gerada pela inclusão desta ferramenta.

2.4 Aprendizado de Máquina

Esta seção tem por objetivo introduzir alguns conceitos necessários à respeito da área de aprendizado de máquina e algoritmos de classificação, pois são tópicos fundamentais para o desenvolvimento desta dissertação.

O aprendizado de máquina é uma área de estudo da computação que consiste em fornecer a computadores a capacidade de aprender a realizar tarefas sem serem explicitamente programados para tanto, através da exposição de dados. Segundo Arthur Samuel, cientista da computação que cunhou o termo, a abordagem de aprendizado de máquina pode reduzir o esforço de programar detalhadamente computadores para resolução de alguns problemas (SAMUEL, 1959). Segundo Haykin (2008), o aprendizado de máquina pode ser dividido em três principais tipos, de acordo com a forma à qual ocorre:

- **Aprendizado supervisionado:** Neste tipo de aprendizado os dados de entrada e os respectivos valores de saída para treinar os modelos são fornecidos antecipadamente. Durante o treinamento, os algoritmos buscam uma função que relacione as entradas com as saídas fornecidas. Além disso, essa função deve ser capaz de generalizar para novos casos de entrada. Neste tipo de aprendizado , os valores de saída no conjunto de dados devem ser conhecidos.

- **Aprendizado não-supervisionado:** Neste tipo de aprendizado somente os da-

dos de entrada são utilizados. Neste caso, os algoritmos buscam agrupar os dados de acordo com as características dos mesmos, visando a detecção de padrões nas informações.

- **Aprendizado por reforço:** Similar ao aprendizado não-supervisionado, no aprendizado por reforço são utilizados apenas os dados de entrada. No entanto, existe um retorno sobre o sucesso do processo de aprendizagem, permitindo que o sistema treine com base ma crítica sobre seu desempenho.

O aprendizado supervisionado vem sendo bastante explorado e validado na área de codificação de vídeo. Nos últimos anos, diversos trabalhos foram propostos apresentando os benefícios da utilização destas técnicas para otimização dos processos de codificação, principalmente com a utilização de classificadores. A tarefa de um classificador é tentar prever a classe de um determinado objeto, representado por uma instância, baseado no valor de seus atributos. O conjunto destes atributos é utilizado pelo classificador para prever, ou classificar, um atributo classe. O resultado desta previsão, por sua vez, será validado posteriormente.

Para construção e teste de um modelo de classificação, o algoritmo utiliza dois subconjuntos de dados extraídos da base de dados original. O primeiro é conhecido como conjunto de treinamento, e é utilizado para construir o classificador. O segundo, conhecido como conjunto de testes, é utilizado para testar a precisão do modelo.

Outra vantagem da utilização de algoritmos de classificação na codificação de vídeo é o *tradeoff* entre complexidade no treinamento e desempenho do classificador. Geralmente, os modelos de classificação ofertam uma boa precisão mesmo em modelos simplificados. Tais características permitem que classificadores sejam implementados mesmo em etapas extremamente complexas de um codificador de vídeo, como na ME (GONÇALVES et al., 2019), nas estruturas de particionamento (AMESTOY et al., 2020), e até mesmo com a etapa de treinamento implementada de maneira *online*, ou seja, durante o processo de codificação (CORREA et al., 2021).

Um dos algoritmos mais conhecidos e utilizados na área de codificação de vídeo são as Árvores de Decisão, que utilizam medidas de organização e entropia para criar regras de classificação. Cada regra é criada de forma a dividir os dados da forma mais organizada possível. Sua principal vantagem é a alta interpretabilidade, visto que os modelos podem ser facilmente representados graficamente, exibindo os critérios que foram utilizados no processo de classificação. Além disso, conforme mencionado anteriormente, a classificação não tem um alto custo computacional, e o método lida bem com atributos de baixa importância. No entanto, as Árvores de Decisão possuem alta variância, sendo muito sensíveis a variações nos dados de treino. Pequenas mudanças nos conjuntos de dados podem levar à criação de modelos muito diferentes. Além disso, este algoritmo é muito suscetível ao sobre ajuste, adaptando-se muito

bem aos dados de treino mas possuindo pouca eficácia na classificação de novas instâncias (HARRINGTON, 2012).

O método de Florestas Aleatórias (do inglês *Random Forests*, RF) (BREIMAN, 2001) foi proposto visando usufruir das vantagens existentes nas Árvores de Decisão, contornando suas limitações. O classificador RF é uma árvore de decisão que utiliza mecanismos de dividir para conquistar para resolver problemas de decisão. A ideia se baseia em que problemas mais complexos podem ser divididos em problemas mais simples de forma recursiva. Essas soluções podem ser combinadas em forma de árvore para gerar uma solução de um problema mais complexo. O algoritmo de RF divide o espaço de amostras em subespaços que são ajustados em diferentes modelos, sendo formalmente estruturado em um grafo acíclico em cada nó, ou em um nó de divisão com dois ou mais sucessores dotado de um teste condicional, ou em um nó folha rotulado com uma nova classe (ZHOU et al., 2014). Sendo assim, um classificador RF constrói diversas Árvores de Decisão que serão usadas para classificar um novo exemplo por um mecanismo de voto majoritário. Cada árvore de decisão usa um subconjunto de atributos selecionados aleatoriamente a partir do conjunto original, contendo todos os atributos.

2.5 Trabalhos Relacionados

Como discutido nas seções anteriores, o VVC trata-se de uma evolução do padrão anterior, o HEVC. Por este motivo, muitas das ferramentas implementadas no padrão antigo estão no VVC. Desta maneira, é possível assumir que os trabalhos focados em ferramentas específicas presentes em ambos os codificadores tem grande potencial de apresentar resultados positivos no VVC. Para melhor organizar a distribuição dos assuntos aos quais os trabalhos relacionados se enquadram, esta seção divide-se em quatro temas principais, sendo: visão geral e comparação entre codificadores, algoritmos para particionamento de blocos, aprendizado de máquina na codificação de vídeo e algoritmos para predição *affine*.

2.5.1 Visão Geral e Comparação Entre Codificadores

Por se tratar de um padrão de codificação recente, é possível ainda encontrar na literatura uma série de trabalhos recentes que buscam estudar o comportamento do VVC, e levantar estatísticas à respeito das etapas de codificação ou das ferramentas implementadas em cada módulo. É o caso de autores já citados nesta dissertação, como Takamura (2019) e Pakdaman et al. (2020).

Além dos autores citados, outros destacam-se na pesquisa sobre o comportamento do novo padrão de codificação, como Mansri et al. (2020) que faz uma extensa avaliação da eficiência de codificação dos mais importantes padrões de codificação atuais.

Além disso, os autores também incluem nas comparações algumas implementações otimizadas de codificadores já existentes, tais como o x264 e o x265, ambas implementações otimizadas dos codificadores H.264/AVC e H.265/HEVC, respectivamente. Nos resultados experimentais apresentados é possível visualizar a superioridade do VVC em relação aos demais codificadores no quesito eficiência de codificação. Em contrapartida, os autores apresentam resultados quanto ao tempo de codificação do VVC em relação aos demais codificadores, que dependendo da configuração escolhida pode alcançar 15 vezes o tempo de codificação do HEVC.

Outra contribuição interessante é a de Cerveira et al. (2020). Neste trabalho os autores apresentam um perfil completo do uso de memória requerido pelo VVC. Além disso, é apresentado a distribuição de acesso à memória somente da etapa de predição interquadros. Segundo os autores, as etapas de predição podem ser responsáveis por até 58% do total de acessos a memória. Por fim, os autores apresentam resultados somente para a etapa de predição *affine*, que pode ser responsável por até 15% do total de acessos a memória.

Por fim, Saldanha et al. (2020) traz uma visão geral à respeito dos trabalhos já publicados sobre designs de hardwares dedicados para a codificação de vídeo no padrão VVC. Com base em oito trabalhos publicados, os autores concluem que o principal desafio para esta geração de codificadores consiste em implementar as novas ferramentas em codificadores com demanda de alta vazão, como em aplicações em tempo real, e a codificação de vídeos com resoluções maiores. Além disso, os autores salientam que, até a data de publicação, não há trabalhos na área destinados à etapa de predição interquadros, nem às ferramentas implementadas nela.

2.5.2 Algoritmos para Particionamento de Blocos

Uma das otimizações mais exploradas em um codificador de vídeo para redução do esforço computacional é a simplificação, de maneira inteligente, dos níveis de particionamento de blocos, uma vez que para cada possível tamanho de bloco, é necessário executar diversas etapas complexas do codificador, como as etapas de predição, por exemplo. Além disso, a nova estrutura de particionamento inserida no VVC (MTT) contribuiu ainda mais para o acréscimo em esforço computacional. Neste sentido, alguns trabalhos já propõem melhorias neste quesito para o VVC, uma vez que é possível alcançar grandes taxas de redução de esforço computacional alterando somente uma etapa de codificação.

Em Zhang et al. (2020), os autores propõem um particionamento de CUs rápida para o modo intra baseado na complexidade e na direção da textura da CU. Com base nestas informações, os autores propõem um esquema capaz de descartar modos de particionamento desnecessários. Por exemplo, se a textura da CU indica uma direção horizontal, o algoritmo permite particionamentos horizontais e evita particio-

namentos verticais, que em teoria, seriam descartados no codificador original. Com estas mudanças os autores conseguem alcançar uma redução de 48,58% no tempo de codificação, com impacto mínimo na eficiência de codificação.

Similarmente, Saldanha et al. (2020) também propõe um esquema de particionamento rápido para o modo intra do VVC. Os autores apresentam um esquema bem definido baseado na variância da CU e no modo de Intra *SubPartition* já implementado no VVC, capaz de ignorar os modos verticais ou horizontais quando estes são menos prováveis de resultar no melhor modo de predição. O esquema proposto é capaz de alcançar uma redução de esforço computacional de 31,41%, com impacto na eficiência de codificação muito similar a Zhang et al. (2020).

Por fim, Fan et al. (2020) também propõem um método de particionamento rápido para o modo intra. Os autores utilizam da similaridade de pixels em áreas homogêneas, sendo assim, a variância de pixels é calculada e serve de referência para um esquema de terminação precoce. Neste esquema, caso tanto a textura horizontal quanto a vertical, obtidas através da aplicação do filtro de Sobel, sejam similares, então o modo QT é diretamente escolhido, e as partições BT e TT são impedidas de serem escolhidas. Com este algoritmo, os autores são capazes de reduzir o tempo de codificação em 49,27%, com um impacto maior na eficiência de codificação, quando comparado aos demais trabalhos que propõem algoritmos para particionamento de blocos.

2.5.3 Aprendizado de Máquina na Codificação de Vídeo

Como comentado na seção 2.4, o uso de técnicas de aprendizado de máquina para otimizar processos da codificação de vídeo é um tema bastante explorado ao longo dos últimos anos.

No padrão HEVC, diversos algoritmos foram propostos utilizando técnicas baseadas em aprendizado de máquina. Em Gonçalves et al. (2019), os autores propõem um esquema de terminação precoce baseado em Árvores de Decisão para a etapa de ME do HEVC. Os autores realizaram inicialmente um processo de mineração de dados, extraindo do codificador milhares de instâncias, ou exemplos, para treinamento do modelo. Uma vez treinado de maneira *offline*[1], o modelo foi implementado no codificador HEVC, e apresentou um excelente *tradeoff* entre redução de tempo e eficiência de codificação.

No padrão VVC ainda é pequeno o número de soluções baseadas em aprendizado de máquina. Amestoy et al. (2019) propõem um esquema para particionamento de blocos no VVC baseado em RFs. O modelo foi treinado de forma *offline*, e com base em informações da CU sendo codifica, é capaz de determinar quais modos da MTT

[1] Treinamento *offline* indica que o processo de treinamento foi realizado antecipadamente, ou seja, havia um conjunto de testes estático e definido para a etapa de treinamento.

são mais prováveis de serem escolhidos. Os resultados experimentais mostraram uma redução de esforço computacional de 30% em média. Também é importante ressaltar que o *overhead* inserido pelos modelos teve impacto mínimo, 0,21% do tempo de codificação. Isto mostra que a utilização de RFs é uma boa estratégia para soluções que necessitam baixa complexidade.

Por fim, (NASIRI et al., 2020) explora o uso de redes neurais convolucionais para melhorar a qualidade de imagem do VVC após o processo de reconstrução do quadro. O modelo proposto auxilia na remoção de artefatos da imagem, atuando como uma espécie de filtro. Para treinamento, além das informações da CU atual, são utilizados também informações de quadros vizinhos como entrada para a rede. O uso de redes neurais no processo de codificação é mais comumente utilizado em etapas onde têm-se o bloco reconstruído para ser usado como entrada do modelo, pois isto evita a complexidade de realizar este processo de maneira desnecessária. Por este motivo, as soluções baseados em redes neurais são menos vistas.

2.5.4 Algoritmos para predição *affine*

Na seção 2.3.3 foi apresentado a evolução dos algoritmos para a compensação de movimento *affine*, desde às suas discussões iniciais em Agui; Okawa; Nakajima (1989) e (SULLIVAN; BAKER, 1991), até a sua implementação no codificador de vídeo VVC em Lin et al. (2018). Ao longo destes anos, diversos trabalhos foram apresentados propondo a implementação de modos de estimação e compensação de movimento *affine* em codificadores de vídeo, porém, somente com à inclusão deste módulo no codificador VVC foi que surgiu a necessidade de otimizar esta etapa de codificação. Por este motivo, ainda há poucos trabalhos na literatura que focam exclusivamente na melhoria desta etapa no VVC.

Os autores em Adhuran et al. (2021) estendem a ideia de estimação de movimento *affine* para a predição intra bloco do codificador VVC. Para isto, é implementado um algoritmo de busca em blocos vizinhos para encontrar os PCs necessários para a aplicação da compensação de movimento *affine*. O fato de não haver quadros de referência para o processo de estimação dos PCs faz com que os autores tenham que definir uma região para executar o algoritmo de busca, uma vez que realizar esta tarefa em todo o quadro é uma tarefa bastante custosa para o codificador. Portanto, a região de busca definida pelos autores engloba as CUs já codificadas das regiões esquerda-cima, cima, direita-cima e esquerda. Ainda para otimizar esta tarefa, é implementado uma cache para manter potenciais candidatos a PC. A cache é capaz de manter 64 possíveis candidatos, e é atualizada sempre que um novo bloco é testado. Desta forma, os autores conseguem alcançar uma eficiência de codificação 2,01% superior ao codificador VVC, com um impacto no tempo de codificação de 159%.

O único trabalho encontrado na literatura com foco na etapa de predição *affine*

no codificador de vídeo VVC é o (PARK; KANG, 2019). Neste trabalho, os autores propõem um método capaz de identificar as etapas redundantes da predição *affine* e ignorá-las durante a execução da ME. Este método é dividido em dois estágios. No primeiro, é avaliado a CU pai que originou a CU atual, e são extraídas algumas informações, tais como o nível de particionamento e o melhor modo de particionamento escolhido. Estas informações alimentam um esquema de terminação precoce definido pelos autores, que decide entre executar a etapa de estimação de movimento *affine*, ou encerrar completamente a execução desta etapa. Caso o esquema decida em continuar a execução da estimação de movimento, o segundo estágio é avaliado. Nele, a direção da predição do quadro de referência com o melhor candidato até o momento é avaliada e, com isto, é possível eliminar os quadros de referência que apontam para direções diferentes.

Os resultados experimentais mostraram que o segundo estágio é capaz de reduzir, em média, somente 4% o tempo de codificação da etapa de predição *affine*. O primeiro estágio, por sua vez, apresentou uma redução de 31% no tempo de codificação da predição *affine*. Quando analisado por completo, é possível observar que o método proposto é capaz de reduzir, em média, o tempo da etapa de predição *affine* em 37%, e o tempo total de codificação em 5%, com uma perda na eficiência de codificação de 0,1%. Ainda, nas conclusões, os autores afirmam que o uso de técnicas de aprendizado de máquina para estes modelos podem ser uma evolução interessante do trabalho. Por se tratar do único trabalho encontrado na literatura com foco na predição *affine* do codificador VVC, este trabalho será usado para comparação com o método proposto nesta dissertação.

2.6 Considerações Finais

Como discutido neste capítulo, o codificador de vídeo VVC foi lançado com uma série de ferramentas capazes de superar os demais codificadores atuais em termos de eficiência de codificação. Entretanto, para tal feito, é acrescido de indesejável impacto no esforço computacional. A etapa de ME, embora ocupando uma fatia menor no tempo de codificação do que em outros codificadores, ainda é a etapa que mais demanda esforço do codificador VVC e, por isto, técnicas que possam diminuir esse custo são desejáveis. A predição *affine*, por sua vez, é a única novidade na ME do VVC em comparação com HEVC. Por se tratar de um método bastante complexo, a predição *affine* não era implementada em codificadores anteriores ao VVC. Alguns trabalhos na área exploraram a compensação e a estimação de movimento *affine* anteriormente, porém, há poucos que propõem otimizações diretas nestas etapas. Em especial, o trabalho de Park; Kang (2019) mostrou que é possível obter resultados desejáveis tanto em redução de esforço computacional quanto em eficiência de co-

dificação, controlando o comportamento da predição *affine*. Além disso, técnicas de aprendizado de máquina podem ser incluídas para uma tomada de decisão mais inteligente na escolha entre executar ou não a etapa de predição *affine*.

Neste sentido, o capítulo 3 irá apresentar uma série de análises sobre o comportamento da ME, com foco na etapa de predição *affine* no codificador de vídeo VVC. As análises apresentadas irão guiar o desenvolvimento de um esquema baseado em aprendizado de máquina, capaz de reduzir o esforço computacional da etapa de predição *affine*.

3 ANÁLISE DO COMPORTAMENTO DA ESTIMAÇÃO DE MOVIMENTO NO PADRÃO VVC

Este capítulo tem como objetivo principal apresentar uma série análises realizadas na etapa de ME do codificador de vídeo VVC com o intuito de compreender o comportamento da etapa de estimação de movimento, e auxiliar no desenvolvimento do esquema para redução de esforço computacional.

Na primeira análise, é avaliada a distribuição de tempo de codificação de cada etapa da ME. Com isto, busca-se identificar o custo de inserir a etapa de predição *affine* no processo de ME. Na segunda análise, é realizado um estudo com foco nas etapas da predição *affine*. Inicialmente é verificado o tempo de codificação para realizar cada uma das etapas (4-parâmetros e 6-parâmetros). Após, é verificada a contribuição de cada etapa na geração dos melhores MVs, isto é, o número de vezes em que a predição *affine* gerou os MVs com menor custo na etapa de ME. Por fim, é realizada uma análise do custo de remover completamente a etapa de predição *affine* do codificador VVC. Com esta análise é possível identificar tanto o custo em eficiência de codificação quanto a redução do tempo de codificação ao remover da ME a etapa de predição affine.

Nas próximas seções é mostrada a metodologia aplicada nos experimentos, e discutido separadamente cada resultado obtido.

3.1 Metodologia de Análise

A metodologia para as análises consiste em realizar modificações no codificador VVC, para coletar, em tempo de execução, os parâmetros necessários para as avaliações. O software de referência do VVC, o VVC Test Model (VTM), na versão 9.0 foi selecionado para os experimentos. Todas as análises foram conduzidas em uma *workstation* com sistema operacional Ubuntu na versão 16.04.7 LTS, processador Intel Xeon Gold 5120 de 14 núcleos e 28 *threads*, e memória RAM de 64GB.

Ao todo, 23 sequências de vídeo foram codificadas, de diferentes resoluções, distribuição de textura e taxa de movimentação. Os vídeos das classes A1, A2, B, C,

D, e F foram selecionados seguindo as recomendações nas Condições Comuns de Teste (CTC, do inglês *Common Test Coditions*) (BOSSEN, 2020). A Tabela 1 lista todas as sequências de vídeo utilizadas para as análises, e suas respectivas classes, resoluções, *framerate* e número de quadros codificados.

Para possibilitar que o ambiente de simulação encontrado neste trabalho possa ser reproduzido em outros cenários, o suporte as instruções SIMD (do inglês *Single Instruction Multiple Data*) foi desabilitado. Além disso, o suporte a codificação *multithreading* também foi desabilitado, sendo assim, cada simulação foi restrita a um único núcleo do processador para codificação. Cada sequência de vídeo foi codificada quatro vezes, cada uma com um Parâmetro de Quantização (QP, do inglês *Quantization Parameter*) distinto, sendo estes 22, 27, 32 e 37.

As principais configurações do codificador VTM, que possibilitam a reprodução das condições deste trabalho, são: configuração temporal *Random Access*, *Search Range* de 384, modo *affine* habilitado.

Tabela 1 – Lista de sequências de vídeos utilizadas e suas respectivas configurações.

Sequência	**Classe**	**Resolução**	***Framerate***	**Quadros**
SlideShow	F	1280×720	20	500
SlideEditing	F	1280×720	30	300
BasketballDrillText	F	832×480	50	500
ArenaOfValor	F	1920×1080	60	64
BasketballPass	D	416×240	50	500
BlowingBubbles	D	416×240	50	500
BQSquare	D	416×240	60	600
RaceHorses	D	416×240	30	300
BasketballDrill	C	832×480	50	500
PartyScene	C	832×480	50	500
BQMall	C	832×480	60	600
RaceHorsesC	C	832×480	30	300
BQTerrace	B	1920×1080	60	64
BasketballDrive	B	1920×1080	50	64
Cactus	B	1920×1080	50	64
RitualDance	B	1920×1080	60	64
MarketPlace	B	1920×1080	60	64
ParkRunning3	A2	3840×2160	50	16
DaylightRoad2	A2	3840×2160	60	16
CatRobot	A2	3840×2160	60	16
Campfire	A1	3840×2160	30	16
FoodMarket4	A1	3840×2160	60	16
Tango2	A1	3840×2160	60	16

3.2 Métricas Objetivas para Avaliação de Desempenho

De forma a avaliar o desempenho das modificações realizadas no codificador VTM, este trabalho utiliza-se das métricas de redução do tempo de execução na codificação e eficiência de codificação dos quadros codificados. Para obter estes resultados, é necessário realizar a comparação com os resultados alcançados pelo VTM sem nenhuma modificação e, portanto, inicialmente é realizada uma etapa de simulações com o VTM original, utilizando as mesmas configurações descritas na seção 3.1. Em seguida, as modificações necessárias são inseridas no codificador VTM, para então ser realizado uma nova etapa de simulação. Por fim, os resultados obtidos nas duas etapas são comparados.

A redução do tempo de execução (RTE) pode ser calculada para todo o processo de codificação, mas também pode ser especializada para um módulo específico. Para isto, é somado o tempo exclusivo para a realização de cada execução do módulo avaliado e, ao final da codificação, estes valores são apresentados. Os resultados de RTE são computados a partir da Equação 6,

$$RTE = \frac{T_O - T_M}{T_O} \times 100 \tag{6}$$

onde T_O representa o tempo de codificação do VTM original e T_M representa o tempo de codificação do VTM modificado.

Os resultados considerando a eficiência de codificação foram obtidos a partir da métrica *Bjontegaard Delta-rate* (BD-rate) (BJONTEGAARD, 2001). Esta métrica é frequentemente utilizada na área de codificação de vídeo e realiza uma relação entre a taxa de bits e a qualidade do vídeo codificado. Visando facilitar a compreensão da métrica BD-rate, define-se BS_O como o *bitstream* gerado pela codificação de uma sequência de vídeo realizada pelo VTM original e BS_M como o *bitstream* gerado pela codificação de uma sequência de vídeo realizada pelo VTM com algum tipo de modificação. Assim, um acréscimo percentual em BD-rate significa o percentual de aumento na taxa de bits de BS_M em relação à BS_O para que ambos vídeos reconstruídos possuam a mesma qualidade objetiva de imagem. Portanto, conclui-se que um aumento em BD-rate significa que as modificações implementadas no VTM geram um impacto negativo na eficiência do codificador.

Com as configurações de codificação e sequências de vídeo utilizadas descritas, e as métricas objetivas definidas, é possível descrever as análises realizadas no codificador VTM, bem como apresentar os resultados obtidos dos experimentos.

3.3 Análise do Impacto da Estimação de Movimento no Codificador VVC

Assim como os demais codificadores de vídeos atuais, o VVC também apresenta uma etapa de estimação de movimento (ME) bastante complexa. Segundo Pakdaman et al. (2020), a ME do VVC pode ser responsável por até 57% do tempo total de codificação, dependendo da configuração utilizada. Embora ainda seja a etapa mais custosa do codificador, a relação da complexidade da ME do VVC é menor em relação a outros padrões de codificação. Segundo Kim et al. (2012), a complexidade da ME no HEVC poderia ser de até 70%. Esta diminuição da relação do tempo de codificação da ME no VVC não significa que esta etapa está menos complexa, pelo contrário, a inclusão de novas ferramentas em outras etapas, conforme mencionado na seção 2.3, tornaram a ME menos expressiva, entretanto, também tornaram o codificador como um todo mais complexo.

Neste sentido, primeiramente é realizada uma análise para medir o custo computacional da etapa de ME no codificador VVC. Para isto, é coletado o tempo de execução da ME para todos os possíveis tamanhos de CUs codificadas. Após o processo de codificação, é possível obter o tempo de execução total da ME para todas as sequências de vídeo selecionadas, cada uma codificada com os quatro valores de QPs descritos anteriormente.

A Figura 12 ilustra a porcentagem da etapa ME em cada classe de sequências de vídeo. Para obtenção destes valores, foi realizada a média de todas as sequências de vídeo pertencente à classe em questão, para cada valor de QP. Sabe-se que quanto menor o QP, mais esforço de codificação é empregado (CHEN, 2020) e, consequentemente, o resultado final é uma codificação com maior qualidade objetiva. Neste sentido, é possível observar que em todos os casos, a porcentagem do tempo de execução da ME cresce a medida que valores de QPs maiores são selecionados. Isto deve-se principalmente à complexidade empregada na etapa de predição intraquadro que, por sua vez, precisa avaliar muito mais blocos candidatos quando um valor de QP menor é selecionado.

Quando verificada cada sequência de vídeo em específico (Apêndice A), é possível verificar que o menor tempo de execução da ME em relação ao restante da codificação foi em *Campfire* com QP 22, apenas 26,59% do tempo total. Por se tratar de um vídeo de resolução 4K, sabendo que apenas 16 quadros são codificados nestes casos (veja a Tabela 1), é possível assumir que esta relação menor seja devido ao grande *overhead* inserido pelo primeiro quadro codificado somente com modo intra. Em contrapartida, o maior tempo de execução da ME ficou por conta da sequência de vídeo *MarketPlace* com QP 37, com 59,36% em relação ao tempo de codificação total. Este valor de máxima supera o encontrado por (PAKDAMAN et al., 2020), porém,

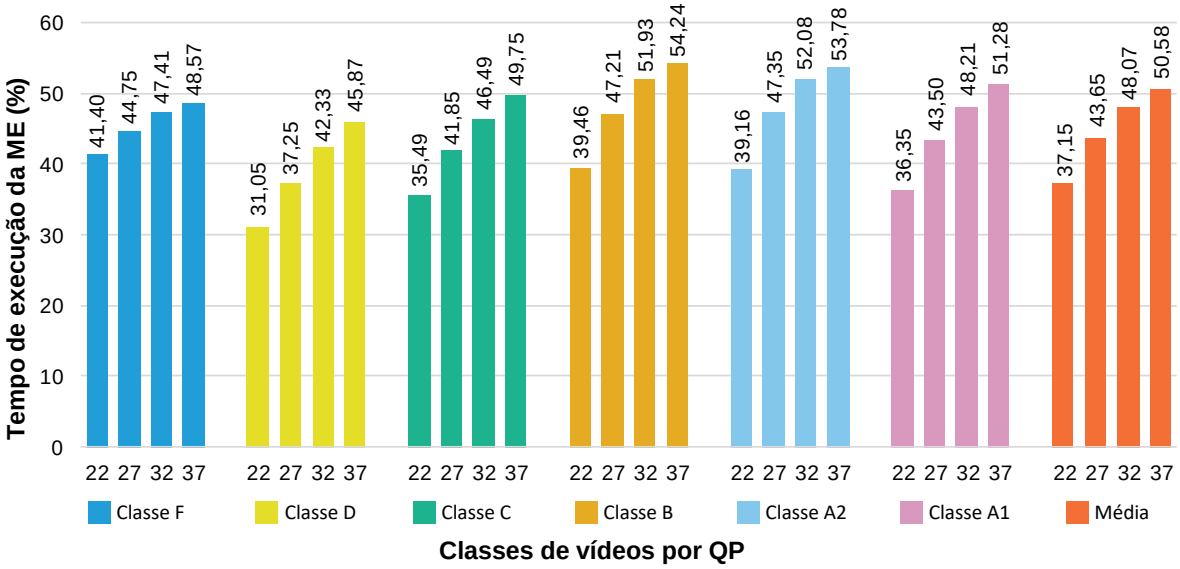


Figura 12 – Tempo médio de execução da ME para cada classe de vídeos.

deve ser levadas em consideração as configurações distintas aplicadas neste trabalho e descritas na seção 3.1.

Quando avaliada a média de todas as classes de sequências de vídeo, na Figura 12, é possível observar que a etapa de ME foi responsável, por 37,15%, 43,65%, 48,07% e 50,58% do tempo total de codificação do VTM, em média, para os valores de QPs 22, 27, 32 e 37, respectivamente. Desta forma, com esta análise é possível verificar que a etapa de ME no codificador VTM tem grande potencial de otimizações que possam reduzir o esforço computacional do processo de codificação. O módulo de ME, por sua vez, ainda é divido em outras etapas, cada uma com suas particularidades que deverão ser levadas em consideração para a definição de uma solução rápida para a ME.

3.4 Análise das Etapas de Codificação da Estimação de Movimento

Conforme mostrado na seção anterior, o módulo de ME do codificador VTM foi responsável por até 59,36% do tempo total de codificação. Entretanto, não se sabe o quanto cada etapa da ME representa deste total. Sendo assim, nesta seção é apresentada uma análise com focos nas etapas de predição unilateral, bilateral e *affine*, que compõem a ME do VVC. Para isto, similarmente a análise anterior, é coletado o tempo de execução de cada etapa separadamente e, ao final, os resultados médios por classe de vídeo são apresentados.

A Tabela 2 mostra os resultados obtidos, separados por valores de QPs. Quando avaliados os resultados médios, é possível identificar que a predição unilateral é a etapa que menos demanda esforço da ME, sendo responsável por cerca de 22% do

tempo total de execução da ME. A etapa de predição bilateral, por sua vez, é responsável por 35%, enquanto que a predição *affine* é a etapa que mais demanda esforço da ME, com 43% do tempo total.

Também é possível perceber que tanto a predição unilateral quando bilateral tendem a diminuir a proporção do tempo de execução da ME à medida que valores maiores de QP são atribuídos, enquanto que, para a predição *affine* a condição contrária é verificada. Isto deve-se, principalmente, à restrição de tamanho de CU mínima para a execução da predição *affine* (veja a Seção 2.3.3). Sendo assim, enquanto que as etapas anteriores são mais requisitadas quando CUs de tamanhos menores são codificadas, a predição *affine* apresenta uma contribuição menor, uma vez que não é processada nestes casos.

Portanto, com esta análise é possível assumir que a inclusão de uma nova ferramenta de predição acarretou em um acréscimo substancial na complexidade da etapa de ME, que já era bastante complexa nos codificadores anteriores. A predição *affine*

Tabela 2 – Tempo de execução da ME por etapas.

Classe de vídeo	**QP**	**Tempo de execução da ME (%)**		
		Unilateral	**Bilateral**	**Affine**
Classe F	22	24	36	40
	27	22	34	44
	32	22	33	45
	37	22	32	46
Classe D	22	26	38	36
	27	23	35	42
	32	20	35	45
	37	20	34	46
Classe C	22	25	38	37
	27	23	35	42
	32	22	34	44
	37	21	33	46
Classe B	22	24	40	36
	27	22	35	43
	32	22	32	46
	37	23	30	47
Classe A2	22	24	36	40
	27	22	34	44
	32	22	33	45
	37	22	32	46
Classe A1	22	26	38	36
	27	23	35	42
	32	20	35	45
	37	20	34	46
Média	**22**	**25**	**38**	**38**
	27	**23**	**35**	**43**
	32	**21**	**34**	**45**
	37	**21**	**33**	**46**

passa a ser a etapa mais complexa da ME, podendo ser responsável por até 47% do tempo total de execução da ME.

3.5 Análises da Etapa de Predição *affine*

Conforme verificado nas análises anteriores, a predição *affine* é a etapa que mais demanda esforço computacional da ME do codificador VVC, podendo ser responsável por até 47% do tempo total de codificação da ME. Contudo, conforme discutido na Subseção 2.3.3, a predição *affine* é dividida em duas etapas: 4-parâmetros e 6-parâmetros. Neste sentido, esta seção tem como objetivo avaliar o comportamento da predição *affine* dentro da ME.

Primeiramente, foi verificada a relação do tempo de execução das etapas de 4-parâmetros e 6-parâmetros da predição *affine*. A Figura 13 mostra a porcentagem de cada etapa no tempo de execução da predição *affine*, onde nas regiões com barras sólidas encontra-se a etapa de 4-parâmetros, enquanto que nas regiões com barras horizontais encontra-se a etapa de 6-parâmetros. Primeiramente, é possível observar a distribuição permaneceu bastante similar para todos os casos. É possível verificar uma crescente na proporção de tempo consumido na etapa de predição 4-parâmetros à medida que os valores de QPs maiores são atribuídos. Isto deve-se principalmente a condição para execução da etapa de 6-parâmetros discutida na Subseção 2.3.3, que verifica se o custo obtido a partir do MV gerado da etapa de 4-parâmetros é até 5% maior que o custo obtido a partir da execução da predição unilateral e bilateral. Desta forma, é possível assumir que, quando codificado com valores de QPs mais altos, a

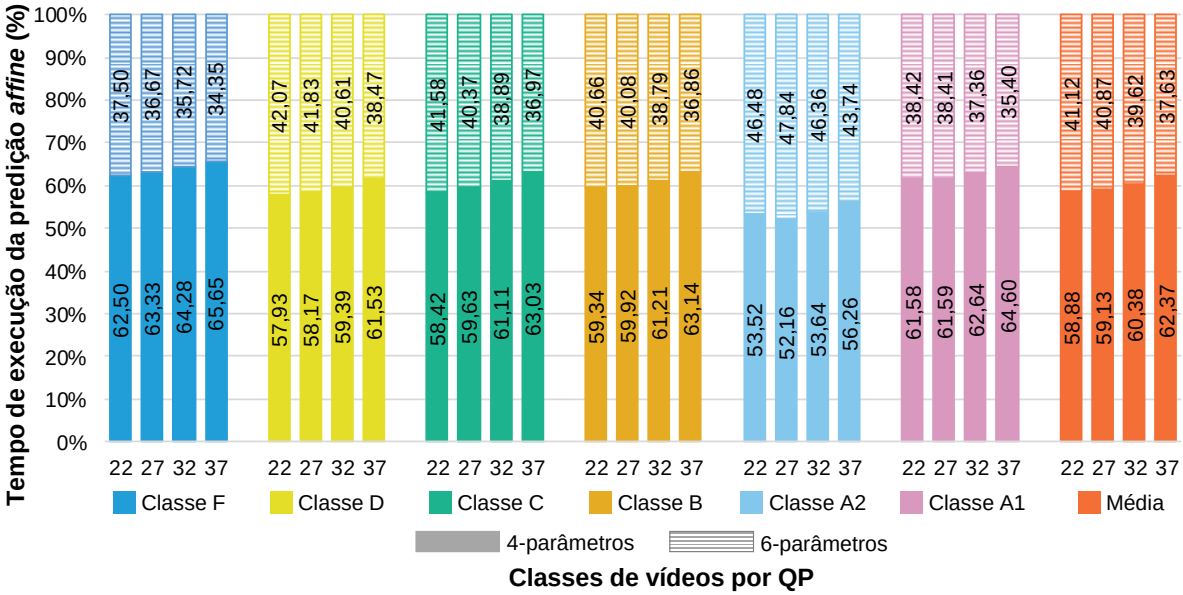


Figura 13 – Relação do tempo médio de execução das etapas de predição *affine* para cada classe de vídeos.

perda de definição, principalmente na região das bordas, afetou a qualidade obtida da predição *affine*.

Ainda quando observamos a Figura 13, é possível verificar que a etapa de 4-parâmetros, mesmo no caso com menor impacto, foi responsável por 52,16% do tempo de execução da predição *affine*, em média para todas as sequências de vídeo da classe A2 QP 27. Isto deve-se ao fato de que não há uma restrição além do tamanho de bloco para o codificador realizar a etapa de 4-parâmetros e, portanto, o número de vezes que o codificador executa esta etapa é maior que a de 6-parâmetros. Por outro lado, a etapa de 6-parâmetros, mesmo no caso com menor impacto, foi responsável por 34,35% do tempo de execução da predição *affine* em média para as sequências de vídeo da classe F com QP 37.

Após isto, foi verificado na Figura 14 o total de vezes em que o codificador executa a etapa de predição *affine* por completo, isto é, processa tanto a etapa de 4-parâmetros quanto a 6-parâmetros. Em azul, é apresentado o número de vezes em que a etapa de 6-parâmetros foi avaliada após a etapa de 4-parâmetros. Nesta análise, é possível perceber que, em todas as sequências de vídeo analisadas, o número de execuções da etapa de 6-parâmetros após a etapa de 4-parâmetros foi pelo menos 80%, em média para os quatro valores de QP. Isto indica que, em pelo menos 80% dos casos, o vetor de movimento encontrado na etapa de 4-parâmetros teve um custo até 5% maior que o melhor vetor encontrado nas etapas de predição unilateral e bilateral. No maior caso, para *SlideEditing*, este comportamento ocorreu em 95,20% dos casos.

Ainda na Figura 14, foi analisado o número de vezes em que a predição *affine* obteve o melhor MV, isto é, o total de vezes em que o vetor encontrado tanto na etapa de 4-parâmetros quanto na etapa de 6-parâmetros apresentou o menor custo de toda a ME. Em laranja, a porcentagem média destes casos é apresentada para

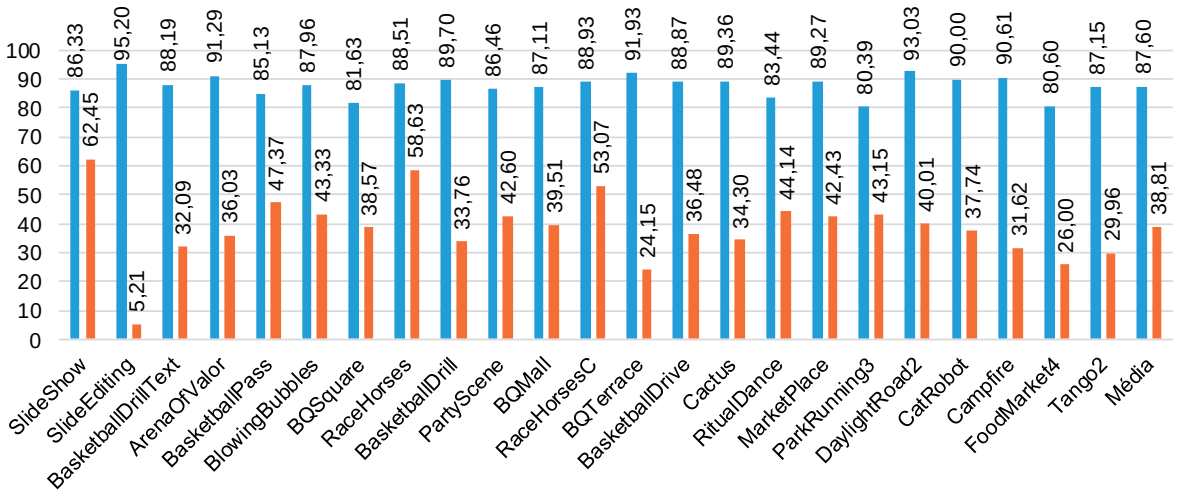


Figura 14 – Porcentagem de execução da etapa de 6-parâmetros em relação à 4-parâmetros (em azul) e porcentagem de vezes em que predição *affine* obteve o melhor MV em toda a ME (em laranja).

cada sequência de vídeo. No melhor caso, para a sequência de vídeo *SlideShow*, em 62,45% dos casos, a etapa de predição *affine* obteve o vetor com menor custo. Isto deve-se, principalmente, a natureza do conteúdo, que trata-se de uma apresentação de *slides*, onde cada transição aplica uma série de transformações na cena. Por outro lado, o pior caso foi observado em *SlideEditing*, com apenas 5,21% dos casos. Novamente, este resultado é devido a natureza do conteúdo da sequência de vídeo, que trata-se de uma captura de edição de texto, onde as únicas transições de objetos são as miniaturas que se movem na vertical, ou seja, um movimento translacional em um único eixo. Desta forma, a predição *affine*, que é responsável pelos movimentos não translacionais, não tem grande impacto no resultado final. Em média, quando processada, a predição *affine* só foi efetiva em 38,81% dos casos, sendo que nos 61,19% restantes o processamento desta etapa foi desperdiçado.

Por fim, foi verificada a eficiência de codificação, em termos de BD-rate, do codificador VTM ao desabilitar por completo a etapa de predição *affine*. A Tabela 3 mostra os resultados obtidos nesta análise. É possível verificar que em *SlideEditing*, o BD-rate foi de 0,0460%. Isto indica que para esta sequência de vídeo a etapa de predição *affine* teve pouco impacto na eficiência de codificação. Tal comportamento já

Tabela 3 – Resultados obtidos da remoção da etapa de predição affine em relação ao codificador VTM original.

Classe	Sequência de Vídeo	RT Total (%)	BD-rate (%)
F	*SlideShow*	10,51	7,3405
	SlideEditing	0,50	0,0460
	BasketballDrillText	3,02	1,0940
	ArenaOfValor	7,30	1,8790
D	*BasketballPass*	2,43	0,9304
	BlowingBubbles	4,45	2,4820
	BQSquare	8,08	2,9565
	RaceHorses	8,78	4,5468
C	*BasketballDrill*	7,13	2,2400
	PartyScene	6,65	2,0020
	BQMall	4,87	1,6903
	RaceHorsesC	8,54	4,8673
B	*BQTerrace*	3,46	1,1080
	BasketballDrive	3,67	1,2100
	Cactus	9,42	3,6429
	RitualDance	6,00	1,3298
	MarketPlace	6,02	3,1023
A2	*ParkRunning3*	4,87	1,5489
	DaylightRoad2	1,14	0,9593
	CatRobot	3,11	1,9840
A1	*Campfire*	1,76	0,5902
	FoodMarket4	2,93	1,1754
	Tango2	2,15	0,6879
-	**Média**	**5,08**	**2,1484**

havia sido evidenciado na análise anterior, uma vez que a predição *affine* apresentou a menor contribuição na escolha dos melhores vetores para esta sequência. Por outro lado, em *SlideShow*, é possível observar que a eficiência de codificação obtida após desabilitar a predição *affine* sofreu uma perda bem mais significativa, com BD-rate de 7,3405%. Novamente, este comportamento também havia sido evidenciado na etapa anterior, onde a maior contribuição da predição *affine* foi para esta sequência de vídeo. Quando verificado o resultado médio para todas as sequências de vídeo, desabilitar a predição *affine* ocasionou uma perda de 2,1484% na eficiência de codificação.

Quando verificada a redução de tempo de codificação após desabilitar a predição *affine*, nota-se que os valores obtidos são distintos do tempo total desta etapa quando verificado nas análises anteriores. Isto deve-se ao motivo que simplesmente desabilitar esta etapa não garante completa eficiência uma vez que, para compensar as perdas obtidas em remover a etapa de predição *affine*, o codificador emprega maior refinamento nas outras etapas de codificação. Ainda assim, os valores obtidos em termos de redução de tempo de codificação foram de até 10,51% em *SlideEditing*, com uma média de 5,08% para todas as sequências.

Sendo assim, as análises apresentadas neste capítulo mostraram que a ME do codificador VTM ainda é bastante complexa, podendo ser responsável por até 59,36% do tempo total de codificação. Neste sentido, a etapa da ME que mais demanda esforço do codificador é a predição *affine*, podendo ser responsável por 43% do tempo total de execução da ME, em média. Além disso, simplesmente desabilitar a etapa de predição *affine* do codificador VTM causou um impacto indesejável na eficiência de codificação de 2,1484%, em média.

De fato, a etapa de predição *affine* é a única adição do padrão VVC em relação ao seu antecessor na etapa de ME e, por isso, ainda não foram exploradas otimizações com foco nesta etapa em nenhum trabalho publicado na literatura científica. Neste sentido, este trabalho propõe, no próximo capítulo, um esquema baseado em aprendizado de máquina, capaz de identificar os casos onde a etapa de predição *affine* pode ser encerrada sem causar grandes penalidades na eficiência de codificação, dado que esta etapa se mostrou bastante custosa para o codificador, mas traz benefícios na eficiência de codificação em alguns cenários. Além disso, o esquema proposto pode ser facilmente combinado com outras técnicas de redução de esforço computacional em outras etapas da codificação, podendo oferecer uma redução de esforço computacional ainda maior.

4 LEARNING-BASED AFFINE PREDICTION

A partir das análises apresentadas no capítulo 3, verificou-se que é interessante e possível interromper a execução da predição *affine* em casos onde o melhor MV já havia sido encontrado nas etapas anteriores, de predição unilateral e bilateral. Sendo assim, este capítulo propõe uma solução para redução do esforço computacional da ME do padrão VVC.

Chamado de LEAP (do inglês *Learning-based Affine Prediction*), o esquema proposto é baseado na exploração de Florestas Aleatórias, e é treinado para identificar os casos onde o melhor vetor de movimento já foi encontrado nas etapas anteriores à predição *affine*. A Figura 15 apresenta um fluxograma das principais etapas da ME, com a inserção do fluxo de execução do LEAP. O primeiro passo do LEAP é realizado antes da etapa de 4-parâmetros da predição *affine* e tem como objetivo decidir se o codificador deve encerrar a execução da ME neste ponto. Caso o modelo decida por continuar, a etapa de 4-parâmetros é processada e o segundo passo é executado. Novamente, o modelo decide se o resultado obtido da etapa de 4-parâmetros é suficiente, ou se deve ser executado a etapa de 6-parâmetros.

Nas próximas seções, será apresentada toda a metodologia para coleta dos parâmetros e construção dos conjuntos de dados para treinamento, o pré-processamento dos dados, o treinamento e otimização dos modelos e, por fim, o processo de implementação dos modelos treinados no codificador VVC.

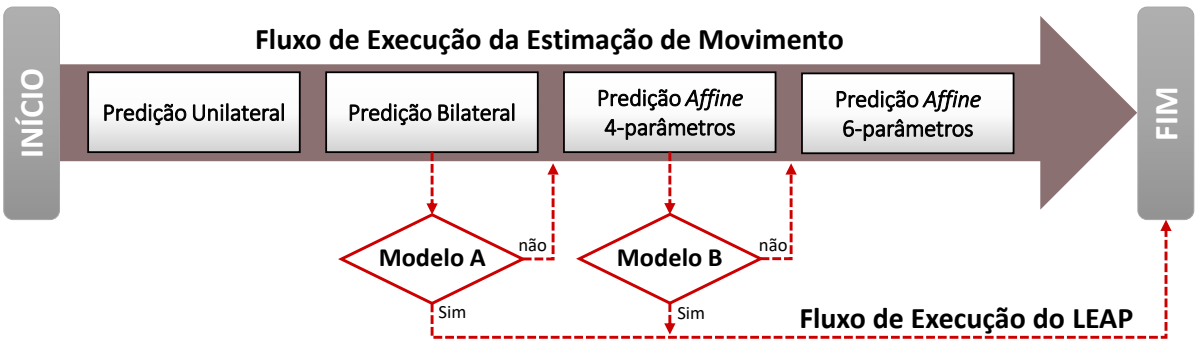


Figura 15 – Fluxograma de execução da ME com o algoritmo LEAP implementado.

4.1 Processo Mineração de Dados e Construção dos Conjuntos de Dados

O processo de mineração de dados consiste em extrair do codificador os parâmetros necessários para treinamento dos modelos. Para definição do LEAP, 33 parâmetros foram coletados durante a execução da ME no software de referência do VVC. Os parâmetros foram organizados em um conjunto de entradas, chamadas de instâncias. Ao todo, 15 sequências de vídeo, selecionadas especificamente para a etapa de treinamento, foram codificadas para obtenção das instâncias necessárias. As instâncias coletadas foram balanceadas e agrupadas em 6 conjuntos de dados distintos. Esta seção descreve com detalhes cada uma destas etapas.

Inicialmente, na etapa de pré-treino, foram consideradas 73 sequências de vídeos para dar início ao processo de coleta de dados. As sequências de vídeo selecionadas para a etapa de pré treino são: 53 do NETVC, 8 do *Ultra Video Group*, e 12 da ex-CTC. A primeira base de dados diz respeito ao *Internet Video Codec* (NETVC), que é um órgão que lança periodicamente documentos abordando recomendações e condições de testes ideias a serem executados entre diferentes codificadores de código aberto (DAEDE; NORKIN; BRAILOVKKIY, 2016). A segunda base de dados tem origem do *Ultra Video Group* (UVG), que é um grupo acadêmico de codificação de vídeo da Finlândia com mais de 20 anos de experiência na condução de pesquisas pioneiras em sistemas de processamento de vídeo e imagem (MERCAT; VIITANEN; VANNE, 2020). A terceira base de dados, diz respeito às sequências de vídeo que já fizeram parte CTC (BOSSEN, 2020) durante o processo de padronização do VVC, mas que foram removidas das versões mais atuais. Desta forma, todas as sequências selecionadas para esta etapa são de bases de dados distintas das apresentadas na seção 3.1 e não fazem parte da CTC atual (BOSSEN, 2020), portanto, não serão utilizadas na etapa de testes.

Entretanto, realizar o processo de mineração de dados em 73 sequências de vídeos distintas é bastante custoso, uma vez que cada uma deve ser codificada quatro vezes utilizando os valores de QPs listados na seção 3.1, totalizando 292 codificações distintas. Considerando que cada um dos processos de codificação para coleta de dados consome cerca de 90 minutos e gera um arquivo de saída de aproximadamente 800 MBs, executar esta etapa para 292 codificações distintas torna inviável o armazenamento, transferência e manipulação dos dados. Uma etapa inicial de seleção de sequências de vídeo foi realizada para reduzir o escopo. Para isto, utilizou-se a técnica analítica de Itu-t (2008), que calcula e relaciona as sequências de vídeo em dois eixos: variabilidade de dados em um mesmo quadro (*Spatial Information* - SI), e a variabilidade de dados ao longo de todos os quadros (*Temporal Information* - TI). As 73 sequências de vídeo foram organizadas em três conjuntos, de acordo com suas

resoluções: 17 de resolução HD, 32 de resolução Full HD e 24 de resolução Ultra HD. Os valores de SI e TI obtidos para cada uma das sequências de vídeo foram normalizados de acordo com seu respectivo conjunto, e dispostos na Figura 16, onde o eixo *x* representa os valores de SI e o eixo *y* representa os valores de TI. Para cada um dos conjuntos, foi calculada a mediana nos eixos *x* e *y*, possibilitando dividir cada conjunto em quatro quadrantes. Com isto, para escolher as sequências de vídeo que mais diferem entre si em cada conjunto, foram selecionadas cinco sequências de vídeo: Quatro mais distantes da mediana em cada quadrante e uma mais próxima do centro (cruzamento das medianas de SI e TI). Com isto, é possível obter uma boa diversidade de informações de entrada, ao mesmo tempo que torna a manipulação dos

Figura 16 – Sequências de vídeo selecionadas para a etapa de treinamento.

dados de saída viável. As sequências de vídeo selecionadas a partir deste processo foram circuladas em vermelho na Figura 16.

Com as 15 sequências de vídeo selecionadas é possível realizar a etapa de mineração de dados. Para isto, o codificador de referência do VVC, o VTM, foi modificado para coletar 33 parâmetros em tempo de execução. A Tabela 4 descreve cada um dos parâmetros escolhidos. Em alguns dos parâmetros listados, há mais de um valor associado. Nestes casos, a coluna de quantidade indica o número de valores associados a este parâmetro. Em *mvUni*, há quatro valores associados ao parâmetro, que representam as coordenadas *x* e *y* após a etapa de predição unilateral do melhor MV encontrado para a Lista L_0 e para a Lista L_1. *mvBi* trata-se do mesmo caso anterior, porém, após a etapa de predição bilateral. *costs* possui três valores associados, que dizem respeito ao custo calculado pelo codificador após as etapas de predição unilateral para a Lista L_0 e Lista L_1, e após a etapa de predição bilateral. *mvAffine* possui seis valores associados, que indicam as coordenadas *x* e *y* de cada um dos três MVs gerados para cada ponto de controle após a etapa de predição *affine*. Este parâmetro,

Tabela 4 – Lista de parâmetros coletados na etapa de mineração de dados.

Parâmetro	**Quant.**	**Descrição**
videoQP	1	Quantization Parameter de entrada
videoWidth	1	Largura do vídeo
videoHeight	1	Altura do vídeo
videoFramerate	1	Framerate do vídeo
imv	1	Adaptative MV Precision
cuWidth	1	Largura da CU atual
cuHeight	1	Altura da CU atual
depth	1	Profundidade total da CU atual
qtDepth	1	Profundidade da árvore quaternária da CU atual
mtDepth	1	Profundidade da árvore multi-type tree da CU atual
mvUni	4	Posições x e y do melhor vetor de cada lista após a etapa de predição unilateral
mvBi	4	Posições x e y do melhor vetor de cada lista após a etapa de predição bilateral
costs	3	Custo das etapas de predição unilateral (Lista 0 e Lista 1) e bilateral
interDir	1	Valor que indica a etapa com menor custo antes da predição affine
distUni	1	Distância euclidiana entre vetores da etapa de predição unilateral
distBi	1	Distância euclidiana entre vetores da etapa de predição bilateral
symMode	1	Valor que indica se a symetricME foi executada
currQP	1	Quantization Parameter do quadro atual
mvAffine	6	Vetores de movimento resultantes da etapa de 4-parâmetos da predição affine
costAffine	1	Custo da etapa 4-parâmetros da predição affine

juntamente com o *costAffine*, só estão disponíveis após a etapa de 4-parâmetros da predição *affine* e, portanto, não podem serem considerados no modelo que decide entre executar ou não a etapa de 4-parâmetros.

Cada um dos parâmetros coletados é escrito em um arquivo de saída após a execução da etapa de ME para cada tamanho de CU. Por se tratar de um aprendizado supervisionado, uma informação alvo deve ser fornecida para cada instância. Para definir o alvo, é verificada qual etapa originou o MV com menor custo ao final da execução da ME. Os valores alvos também são chamados de classes, e são definidos por: (*before*) Indica que o vetor com menor custo foi obtido antes da etapa de predição *affine*; (*4param*) Indica que o vetor com menor custo foi obtido após a execução da etapa 4-parâmetros da predição *affine*; (*6param*) Indica que o vetor com menor custo foi obtido após a execução da etapa 6-parâmetros da predição *affine*.

Coletar dados da execução da ME para todos os possíveis tamanhos de CU ocasiona em um alto volume de dados e muitas instâncias repetidas. Devido a esta grande quantidade de dados gerados, foram codificados apenas os 16 primeiros quadros de cada uma das 15 sequências de vídeo. Cada sequência foi codificada quatro vezes, com os quatro valores de QPs definidos na seção 3.1. As demais configurações descritas na metodologia também se aplicam para esta etapa. As instâncias de saída foram agrupadas em 12 conjuntos que representam os possíveis tamanhos de CU para a predição *affine*, que são: 128×128, 128×64, 64×128, 64×64, 64×32, 32×64, 32×32, 64×16, 16×64, 32×16, 16×32, 16×16.

Pela natureza dos dados, o número de instâncias geradas para CUs maiores é menor que para CUs menores. Por exemplo, o número de instâncias geradas para as CUs de tamanho 128×128 é muito inferior às CUs 64×64. No ramo de aprendizado de máquina este fenômeno é conhecido como desbalanceamento de dados. Realizar o processo de treinamento para dados desbalanceados pode ocasionar em diversos problemas, fazendo com que os modelos treinados deixem de ser genéricos, e se especializem em casos com maior ocorrência. Para contornar este problema, inicialmente foi decidido dividir cada conjunto de dados possuindo três classes de decisão (*before*, *4param* e *6param*) em outros dois conjuntos A e B com decisões binárias. As decisões binárias são definidas por: (0) Indica que a execução não deve ser interrompida neste ponto, e (1) Indica que a execução deve ser interrompida. Isto significa que no primeiro conjunto de dados A, a classe *before* é caracterizada pela decisão 1, pois indica que o melhor vetor de movimento já foi encontrado nas etapas anteriores à predição *affine*. Por outro lado, as classes *4param* e *6param* devem ser agrupadas e classificadas como decisão 0, pois indicam que o melhor vetor de movimento deve ser encontrado nas etapas de predição *affine*. O segundo conjunto binário B é executado após a etapa de 4-parâmetros, portanto, para estes casos, as classes *before* e *4param* podem serem agrupadas para compor a decisão 1, enquanto que

a classe *6param* define que a decisão 0 deve ser utilizada. Desta forma, é possível reaproveitar instâncias em ambos modelos A e B, podendo assim obter uma maior amostragem, principalmente para o conjunto de dados de CU 128×128, que possui poucos exemplos.

Após aplicar esta técnica, os 12 conjuntos de dados passaram a se tornar 24 conjuntos. Com isso, é possível aplicar a etapa de limpeza e preparação dos dados para o treinamento. Inicialmente, foram removidos os exemplos duplicados para todos os conjuntos e, em seguida, para cada um dos 24 conjuntos foi aplicado a técnica de *downsampling*, que consiste em remover instâncias a fim de que todos os conjuntos tenham o mesmo número de exemplos. Por possuir o menor número de exemplos, o conjunto 128×128 foi utilizado como referência para realizar o *downsampling* de todos os conjuntos. Neste sentido, todos os conjuntos tiveram o número de exemplos reduzidos para 500 para cada classe de decisão, ou seja, para cada uma das 15 sequências de vídeo e para cada um dos quatro valores de QP, foram selecionados 1000 exemplos, sendo 500 classe 0 e 500 da classe 1. Ao final, cada conjunto de dados foi reestabelecido com um total de 60.000 exemplos únicos dos dados da execução da etapa de ME para cada possível tamanho de bloco.

Com os conjuntos de dados balanceados, é possível dar início a etapa de treinamento. Entretanto, é inviável realizar o treinamento em 24 conjuntos de dados distintos, pois torna custoso executar as etapas posteriores de otimização e implementação de cada modelo no codificador VVC. Para diminuir este escopo, foi realizado uma etapa de agrupamento de conjuntos de dados. Nesta tarefa, foi calculado o índice de *feature importance* (PEDREGOSA et al., 2011), que atribui uma pontuação aos parâmetros de entrada dos modelos com base nas suas utilidades na previsão da classe alvo. Os resultados médios obtidos em termos de *feature importance* para todos os conjuntos de dados são apresentados na Figura 17, onde para cada um dos modelos são mostrado os valores dos três parâmetros com maior *feature importance*, o parâmetro com menor, e o parâmetro da mediana, em média para todos os conjuntos. Os valores de *feature importance* para todos os 33 parâmetros avaliados é mostrado no Apêndice C. Com base nesta análise nos 24 conjuntos de dados, foi possível observar uma variação mínima dos parâmetros inferiores a mediana. Entretanto, quando avaliado os parâmetros com maior *feature importance*, é possível perceber um padrão na distribuição. Para os conjuntos de dados 16×16, 16×32, 32×16, 16×64, 64×16 e 32×32, o parâmetro que apresentou maior *feature importance* foi *CostUni* da Lista 0, enquanto que para os conjuntos de dados 32×64, 64×32, 64×64, 64×128 e 128×64, o parâmetro com maior *feature importance* foi o *CostUni* para a Lista 1. Por fim, o conjunto 128×128 foi o único que apresentou o *CostBi* como parâmetro com maior *feature importance*.

Desta forma, os conjuntos de dados foram agrupados seguindo o padrão visuali-

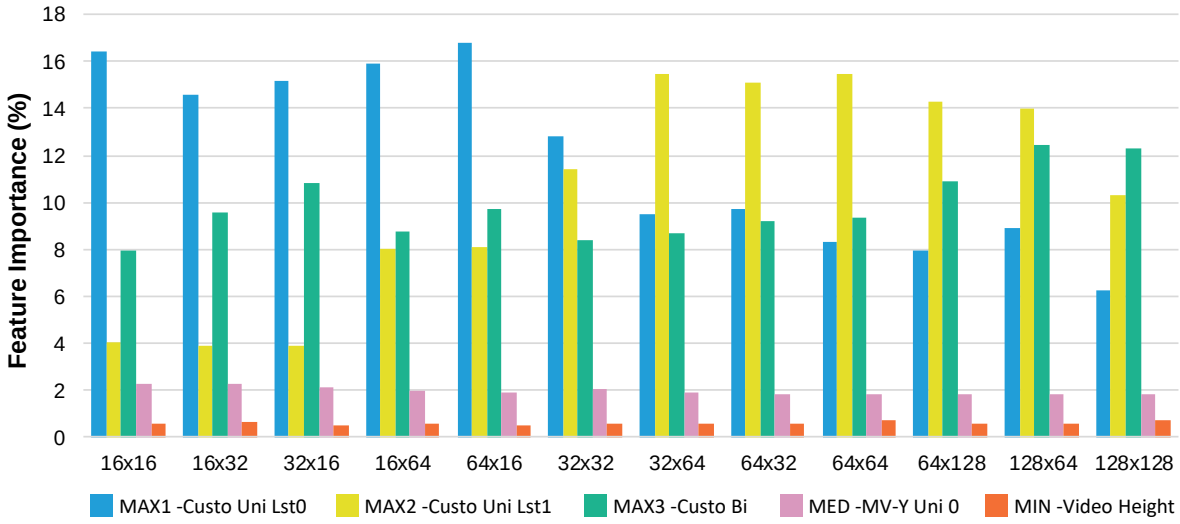


Figura 17 – Valores de *Feature importance* médios obtidos para todos os conjuntos de dados.

zado na análise anterior. Com isto, o número de conjuntos de dados foi reduzido de 24 para apenas 6. A Tabela 5 resume os conjuntos agrupados, bem como seu número final de instâncias, sendo os conjuntos 1A e 1B referente ao primeiro conjunto de dados para os modelos A e B apresentados na Figura 15 e, similarmente, os conjuntos 2A, 2B, 3A e 3B, referentes ao segundo e terceiro conjuntos de dados, também para os modelos A e B, respectivamente. Com base nos seis conjuntos gerados, é possível realizar o treinamento dos modelos com garantia de que os dados estão limpos de duplicações, balanceados e agrupados de acordo com suas similaridades.

4.2 Definição dos Modelos de Aprendizado de Máquina

Com a etapa de mineração para construção dos conjuntos de dados finalizada, é possível descrever o fluxo de treinamento e otimização dos modelos de decisão propostos no esquema LEAP. Neste sentido, esta seção descreve os testes que motivaram a escolha do algoritmo de Florestas Aleatórias para o treinamento dos modelos, bem como a metodologia utilizada para o treino, e a otimização dos parâmetros de configuração.

Conforme mencionado na seção 2.4, o algoritmo de Florestas Aleatórias é um conjunto de Árvores de Decisão com um mecanismo de voto majoritário. Entretanto, a complexidade para treinamento e implementação dos modelos é muito maior que

Tabela 5 – Descrição dos conjuntos de dados após etapa de agrupamento.

Modelo	Conjuntos de dados	Número de instâncias
1A e 1B	16×16, 16×32, 32×16, 16×64, 64×16, 32×32	360.000
2A e 2B	32×64, 64×32, 64×64, 64×128, 128×64	300.000
3A e 3B	128×128	60.000

apenas utilizar uma única árvore de decisão. Além disso, dependendo do problema a ser solucionado, utilizar um modelo de Árvores de Decisão em vez de um conjunto de modelos, pode alcançar uma melhor precisão. Neste sentido, foi realizado uma etapa de validação do uso de Florestas Aleatórias, onde cada modelo foi gerado utilizando os algoritmos de Florestas Aleatórias e Árvores de Decisão. Os algoritmos para treino escolhidos foram o *DecisionTreesClassifier* e o *RandomForestsClassifier*, ambos implementados na biblioteca Scikit-Learn (PEDREGOSA et al., 2011). Os parâmetros de configuração da etapa de treino, também chamados de hiperparâmetros, foram definidos utilizando os valores padrões implementados na biblioteca a saber:

- **DecisionTreesClassifier:** *criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1*
- **RandomForestsClassifier:** *n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1*

, onde *criterion* indica a função para medir a qualidade de uma divisão, *splitter* indica a estratégia usada para dividir cada nodo, *max_depth* indica a profundidade máxima da árvore, *min_samples_split* indica o requisito mínimo de amostras para dividir um nodo interno, *min_samples_leaf* indica o requisito mínimo de amostras para que se tenha um nodo folha, e *n_estimators* indica o número de árvores na floresta.

Tanto para esta etapa de treinamento quanto para as demais, a técnica de reamostragem utilizada foi a validação cruzada de 10. Esta técnica divide o conjunto de dados em 10 iterações, onde em cada, um subconjunto diferente é selecionado para servir como conjunto de testes, e o restante é utilizado como conjunto de treinamento. Desta forma, é garantido que ao final da operação, todo o conjunto de dados foi testado. Ao final da validação cruzada, é informada uma lista de valores contendo a acurácia do modelo em cada iteração. Sendo assim, a Figura 18 apresenta o resultado médio de cada iteração para cada um dos modelos definidos neste trabalho. Quando verificado os resultados obtidos, em todos os modelos o algoritmo de Florestas Aleatórias apresentou maior acurácia que as Árvores de Decisão. A maior acurácia obtida foi no modelo 2A, com 75,12%. Neste mesmo caso, a acurácia obtida no modelo com Árvores de Decisão foi de 70,28%. A menor acurácia obtida foi no modelo 3B, onde o algoritmo de Árvores de Decisão apresentou uma acurácia de 60,56%, enquanto que as Florestas Aleatórias tiveram acurácia de 69,15% neste mesmo modelo. Por fim, quando verificado, a média para as acurácias obtidas em todos os modelos de Florestas Aleatórias mostraram-se superiores as das Árvores de Decisão, com uma média de 72,48%, contra apenas 67,16% obtido com o algoritmo de Árvores de Decisão. Esta análise valida a escolha das Florestas Aleatórias como algoritmo de classificação escolhido para o LEAP.

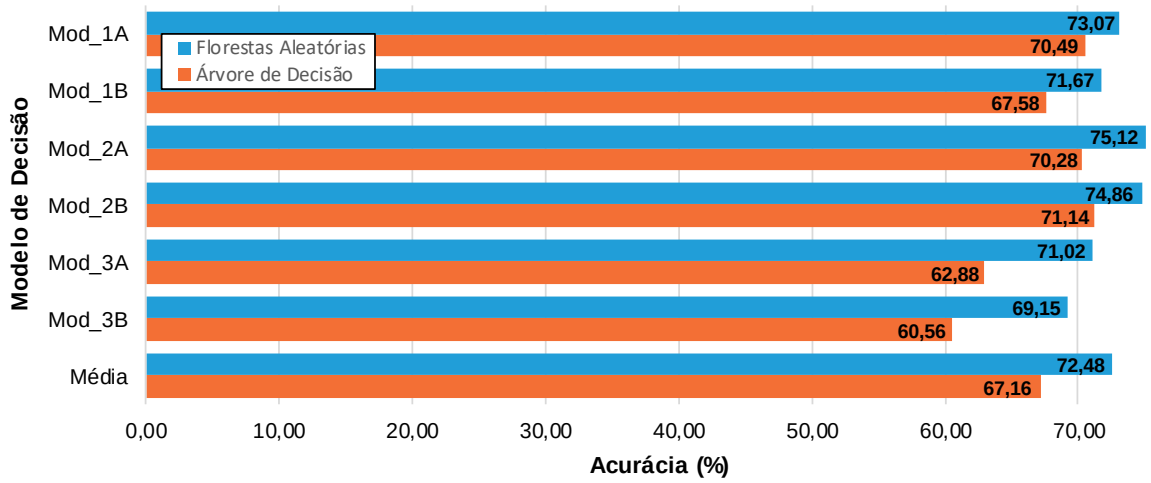


Figura 18 – Comparação dos modelos do LEAP treinadas com algoritmo de Árvores de Decisão e Florestas Aleatórias.

4.3 Treinamento dos Modelos de Florestas Aleatórias

Os modelos apresentados na análise anterior foram obtidos através do algoritmo de treinamento de Florestas Aleatórias utilizando os valores padrões para os hiperparâmetros. Entretanto, resultados superiores podem ser obtidos manipulando os hiperparâmetros da melhor maneira possível. Sendo assim, foi realizado uma estratégia para otimização dos hiperparâmetros, baseado nos ajustes com *Random Search* e *Grid Search*. Ambas estratégias baseiam-se em realizar o treinamento e avaliar o resultado obtido em um número específico de iterações. A estratégia de *Random Search* consiste em selecionar, aleatoriamente, em um intervalo de valores informado para os hiperparâmetros, em um número específico de iterações. Cada iteração passa pelo treinamento e a acurácia final é mantida. Ao final de todas as iterações é fornecida uma matriz contendo os valores escolhidos para os hiperparâmetros e a acurácia obtida em cada iteração. Desta forma, com um número grande de iterações, é possível identificar regiões com maior probabilidade de conter os hiperparâmetros com maior acurácia. A estratégia de *Grid Search* é similar à anterior, porém, avalia todas as combinações possíveis entre os intervalos de hiperparâmetros informado. Devido a este comportamento exponencial, a complexidade para realizar esta técnica é alta e, portanto, não deve ser feita para um grande conjunto de hiperparâmetros.

Na etapa de *Random Search*, é necessário informar um intervalo a qual o algoritmo vai sortear valores para o treinamento e avaliação. Sendo assim, os intervalos de entrada para esta técnica foram: Limite máximo de 100 para *n_estimators*, *min_samples_split* entre 8 e 50, *min_samples_leaf* entre 1 e 30, *max_depth* entre 2 e 20, e *max_samples* entre 0,1 e 1,0. Os melhores resultados obtidos para cada um dos modelos após as 1000 iterações são apresentados na Tabela 6. Além disso, é apresentado o *f1_score* obtido no treinamento com os melhores hiperparâmetros. O

f1_score é uma métrica que combina a precisão do modelo com a frequência em que o classificador encontra os exemplos de uma classe, e é o modelo de saída recomendado neste tipo de abordagem (PEDREGOSA et al., 2011).

Dada a natureza exponencial do número de operações necessárias para realizar uma busca exaustiva em torno dos melhores pontos encontrados na etapa de *Random Search*, é inviável realizar este processo para todos os hiperparâmetros. Por este motivo, para o desenvolvimento do trabalho, foi necessário escolher os hiperparâmetros que mais contribuem para a precisão dos modelos, e iterar em torno deles. Para esta tarefa, foi calculado o coeficiente de correlação de Pearson (CCR) (KIRCH, 2008), que mede a relação estatística entre duas variáveis, para todos os hiperparâmetros em relação ao *f1_score*. Os resultados obtidos nesta etapa são apresentados na Tabela 7, onde para cada hiperparâmetro em cada um dos modelos, um valor que varia de -1 a 1 é atribuído. Neste sentido, um CCR com valor próximo a 0 não é interessante, pois indica que há pouca associação entre as duas variáveis. Por outro lado, quanto mais próximo o CCR estiver de -1 ou de 1, mais associação as variáveis possuem, sendo que um valor próximo a -1 indica que um dos valores aumenta, o outro valor diminui, e um valor próximo a 1 indica que a medida que um dos valores aumenta, o outro aumenta também. Com os resultados do CCR é possível traçar, para cada modelo, os dois hiperparâmetros que mais se relacionam ao *f1_score*. Os hiperparâmetros com maior CCR estão demarcados com fundo cinza na Tabela 7. Para os modelos 1A, 1B, 2A, 3A e 3B, os hiperparâmetros que obtiveram maior CCR foram o *max_depth* e o *n_estimators*, sendo que este último foi superado pelo *min_samples_split* no modelo 2B.

Tabela 6 – Resultados de melhores hiperparâmetros e *f1_score* obtidos após aplicação da técnica *Random Search* com 1000 iterações para cada um dos modelos.

Hiperparâmetros	**Conjuntos de dados**					
	1A	**1B**	**2A**	**2B**	**3A**	**3B**
n_estimators	60	40	80	100	15	18
min_samples_split	28	40	22	28	12	20
min_samples_leaf	5	5	1	3	25	13
max_samples	0,5	0,5	0,9	0,2	0,1	0,2
max_depth	20	18	16	20	16	17
f1_score	**0,738**	**0,725**	**0,768**	**0,753**	**0,724**	**0,706**

Tabela 7 – Coeficiente de correlação de Person obtido para cada hiperparâmetro.

Hiperparâmetros	**Coeficiente de Correlação de Pearson**					
	1A	**1B**	**2A**	**2B**	**3A**	**3B**
n_estimators	0,2340	0,1281	0,1095	0,0690	0,2881	0,1889
min_samples_split	-0,0147	-0,1107	0,0332	0,1306	-0,0149	-0,0202
min_samples_leaf	0,1097	-0,0413	0,0261	-0,0169	0,0752	0,0744
max_samples	0,0423	-0,0130	0,0120	-0,0199	-0,0222	0,0180
max_depth	0,7836	0,8351	0,7527	0,8045	0,5724	0,6442

Com os hiperparâmetros que mais se associam ao *f1_score* selecionados em cada modelo, é possível realizar a etapa de *Grid Search*. Nesta etapa, uma busca exaustiva é feita, testando todas as possíveis combinações de hiperparâmetros fornecidos. Para cada modelo, os dois hiperparâmetros com maior CCR foram selecionados, com um intervalo de 20 valores em torno do melhor caso obtido na Tabela 6. Desta forma, considerando os 21 possíveis valores diferentes em cada um dos hiperparâmetros com maior CCR, para a etapa de *Grid Search*, um total de 441 modelos foram novamente treinados e avaliados. Por fim, os melhores resultados obtidos em termos de *f1_score* foram selecionados e são apresentados na Tabela 8. Quando comparado os valores de *f1_score* obtidos nesta etapa com os obtidos na etapa de *Random Search*, é possível perceber uma otimização em todos os modelos treinados.

Por fim, os hiperparâmetros obtidos ao final da aplicação da técnica *Grid Search* mostrados na Tabela 8 foram utilizados para obter os resultados finais de acurácia. Cada modelo foi novamente treinado utilizando a validação cruzada em 10, e os resultados médios de acurácia obtidos foram de 77,80%, 79,77%, 83,35%, 82,84%, 80,97% e 80,07% para os modelos 1A, 1B, 2A, 2B, 3A e 3B, respectivamente. Neste ponto, é interessante avaliar a distribuição dos casos de falso positivo e falso negativo para cada modelo. Neste problema, um falso positivo indica os casos onde o modelo decide por continuar a execução das etapas, entretanto o melhor vetor já havia sido encontrado anteriormente. Neste caso, não há perda em eficiência de codificação, apenas o modelo deixa de fornecer os ganhos desejados em redução de tempo de codificação. Por outro lado, os falsos negativos indicam os casos onde o codificador decide por encerrar a execução da etapa, sendo que o melhor vetor ainda não havia sido encontrado. Diferentemente do caso anterior, nos falsos negativos há uma perda na eficiência de codificação, caracterizando um erro do modelo. Desta forma, é possível considerar somente os falsos negativos como erros, de fato, e obter uma acurácia real dos modelos de 89,49%, 91,03%, 91,82%, 93,55%, 90,37% e 91,70% para os modelos 1A, 1B, 2A, 2B, 3A e 3B, respectivamente. Se considerarmos as médias dos seis modelos, é possível concluir que o LEAP apresenta uma acurácia média de 80,80%, e uma acurácia real de 91,33%, isto é, considerando os falsos positivos como acertos.

Tabela 8 – Resultados de melhores hiperparâmetros e *f1_score* obtidos após aplicação da técnica *Grid Search* para cada um dos modelos.

Hiperparâmetros	**Conjunto de dados**					
	1A	**1B**	**2A**	**2B**	**3A**	**3B**
n_estimators	60	44	80	100	65	73
min_samples_split	28	40	22	18	12	20
min_samples_leaf	5	5	1	3	25	13
max_samples	0,5	0,5	0,9	0,2	0,1	0,2
max_depth	23	22	22	22	17	17
f1_score	**0,744**	**0,740**	**0,782**	**0,777**	**0,736**	**0,713**

Com os resultados obtidos nas etapas anteriores, os modelos foram treinados pela última vez, desta vez utilizando 100% do conjunto de dados para treinamento, ou seja, sem a realização da etapa de validação cruzada. Com isto, um número maior de exemplos pode ser fornecido para o algoritmo de treinamento, uma vez que não é necessário remover parte do conjunto de dados para testes e validação. Os modelos obtidos foram então adaptados e implementados no codificador VTM para poderem ser utilizados durante o processo de codificação. Sendo assim, no próximo capítulo serão discutidos os resultados experimentais obtidos da implementação do esquema LEAP proposto no software de referência do VVC.

5 RESULTADOS EXPERIMENTAIS

Este capítulo apresenta os resultados experimentais obtidos a partir da implementação do LEAP no codificador de referência do VVC. O esquema proposto, baseado em Florestas Aleatórias, tem como objetivo reduzir o esforço computacional do módulo de ME do VVC, com baixas penalidades na eficiência de codificação, a partir do controle do fluxo de execução da etapa de predição *affine*.

A metodologia usada para obtenção dos resultados é a mesma já apresentada na seção 3.1. Nesta etapa, os modelos de Florestas Aleatórias foram implementadas no codificador VTM, e as simulações foram conduzidas para as 23 sequências de vídeo presentes na CTC (BOSSEN, 2020), que não foram consideradas no processo de treinamento dos modelos de Florestas Aleatórias. Considerando os quatro valores de QP utilizados, ao todo, foram realizadas 92 codificações. Os resultados obtidos são medidos em termos de redução do tempo em todo o processo de codificação, na etapa de ME e na etapa de predição *affine*.

A Tabela 9 apresenta para cada sequência de vídeo individualmente, os resultados médios obtidos na codificação do LEAP com os quatro valores de QPs. Além disso, também é calculado a média para cada classe de vídeo.

Quando avaliado em termos de redução de tempo total do codificador de vídeo VTM, o LEAP foi capaz de alcançar uma redução média de 8,49%. É evidente que as sequências de vídeo que mais se favorecem com a predição *affine* apresentarão maior redução de tempo, uma vez que o LEAP favorece estes casos. É o caso de *FoodMarket* que obteve uma redução de 15,45% no tempo total de codificação, onde a câmera transita por entre as prateleiras do mercado. Neste caso, as diferenças de posição entre os objetos são favoráveis as transformações suportadas na predição *affine*. Similarmente, em *RitualDance* a redução obtida também foi alta, com 13,98% em média. Nesta cena há um casal dançando próximo a câmera, desta forma, os próprios movimentos naturais da dança já caracterizam uma série de transformações não-translacionais, como a rotação e o cisalhamento. Por outro lado, em sequências de vídeo sem objetos com formas bem definidas ou com muitas partículas, a predição *affine* tende a ter menos impacto. É o caso de *Campfire*, onde há pessoas mais dis-

Tabela 9 – Resultados obtidos da implementação do algoritmo LEAP proposto em relação ao codificador VTM original.

Classe Resolução	Vídeo	RT Total (%)	RT ME (%)	RT Affine (%)	BD-rate (%)	Custo dos Modelos (%)
F	§ *SlideShow*	9,82	19,00	34,95	0,2277	0,08
* 832x460	§ *SlideEditing*	8,10	23,85	87,16	0,0356	0,03
§ 1280x720	* *BasketballDrillText*	8,61	20,19	53,43	0,1833	0,07
† 1920x1080	† *ArenaOfValor*	10,60	23,39	55,81	0,2829	0,08
D 416x240	*BasketballPass*	5,21	13,61	36,00	0,1902	0,07
	BlowingBubbles	7,75	21,12	50,18	0,1571	0,07
	BQSquare	8,98	24,51	54,23	0,2390	0,08
	RaceHorses	4,74	12,03	28,84	0,3617	0,08
C 832x480	*BasketballDrill*	8,72	20,15	52,17	0,2632	0,07
	PartyScene	7,01	18,62	44,12	0,2128	0,08
	BQMall	8,32	19,23	46,79	0,1705	0,08
	RaceHorsesC	5,40	12,32	28,56	0,2471	0,08
B 1920x1080	*BQTerrace*	8,04	19,53	54,27	0,0550	0,08
	BasketballDrive	7,65	16,85	42,83	0,1594	0,07
	Cactus	7,45	17,32	43,28	0,0328	0,08
	RitualDance	13,98	26,79	61,68	0,8780	0,08
	MarketPlace	10,73	20,48	44,32	0,2265	0,08
A2 3840x2160	*ParkRunning3*	5,72	14,24	29,33	0,1559	0,09
	DaylightRoad2	7,24	14,58	27,52	0,0413	0,09
	CatRobot	12,14	25,66	48,63	0,1756	0,09
A1 3840x2160	*Campfire*	3,47	11,07	37,93	-0,0080	0,05
	FoodMarket4	15,45	31,04	71,82	-0,0648	0,08
	Tango2	10,04	20,73	45,80	-0,0349	0,08
F	Média da Classe	9,28	21,61	57,84	0,1824	0,07
D	Média da Classe	6,67	17,82	42,32	0,2370	0,07
C	Média da Classe	7,36	17,58	42,91	0,2234	0,08
B	Média da Classe	9,57	20,20	49,28	0,2703	0,08
A2	Média da Classe	8,37	18,16	35,16	0,1243	0,09
A1	Média da Classe	9,65	20,95	51,85	-0,0359	0,07
-	**Média Global**	**8,49**	**19,41**	**46,94**	**0,1821**	**0,07**

tantes da câmera, localizadas atrás de uma fogueira. Desta forma, a predição *affine* tem dificuldade de definir as formas, e mapear os objetos nos quadros de referência. Sendo assim, o resultado em termos de redução de tempo total obtido para esta sequência de vídeo foi a pior em todo o conjunto de teste, com uma redução de apenas 3,47%. Similarmente, cenas com muita movimentação tendem a deformar demais as bordas dos objetos em cena, é o caso de *RaceHorses*, que apresenta uma corrida de cavalos com a câmera em distância média da cena. Como a movimentação é alta, a predição *affine* tende a ter pouca eficácia, desta forma, a redução de tempo total obtida para este caso foi de apenas 4,74%, em média. Quando observado as médias para cada classe de vídeos, é possível perceber que em geral, o esquema proposto beneficia mais os vídeos de maior resolução. Isto deve-se ao nível de detalhamento

maior para estes casos, aliado ao fato de que a predição *affine* só é executada para CUs maiores que 8×8. Isto faz com que os vídeos de menor resolução sejam afetados, uma vez que o particionamento menor tende a ser escolhido mais vezes. A classe A1, obteve a maior redução no tempo de codificação, com 9,65%, em média.

É natural assumir que as sequências de vídeo que apresentaram maiores e menores redução de tempo total mantiveram suas posições, quando observada a redução de tempo total para a ME, e somente para a etapa de predição *affine*. Neste sentido, destaca-se novamente a sequência de vídeo *FoodMarket*, que obteve uma redução média de 31,04% e 71,82% para o tempo de codificação da ME e da etapa de predição *affine*, respectivamente. Quando observado em termos de redução de tempo na etapa de predição *affine*, *SlideEditing* apresentou a maior redução, com 87,16%, em média. Entretanto, assim como observado na análise de esforço computacional empregado em cada etapa, a predição *affine* mostrou-se a etapa menos relevante para esta sequência de vídeo. Sendo assim, este comportamento foi refletido nos resultados experimentais, uma vez que, na grande maioria dos casos, o LEAP optou por não executar a etapa de predição *affine*. Por este motivo, mesmo apresentando a maior redução no tempo de execução da etapa de predição *affine*, *SlideEditing* não apresentou as maiores reduções em termos de tempo de codificação total e da ME. Em média, o LEAP foi capaz de reduzir o tempo de execução da ME e da etapa de predição *affine* em 19,41% e 46,94%, respectivamente.

Os resultados em termos de eficiência de codificação mostraram que o LEAP causou um impacto baixo em relação ao VTM original, com um acréscimo de apenas 0,1821% no BD-rate. Isto indica que o LEAP emprega um aumento na taxa de bits de apenas 0,1821% para manter a mesma qualidade visual que o codificador VTM original. O pior caso em termos de eficiência de codificação foi para a sequência de vídeo *RitualDance*. Assim como verificado anteriormente, está é uma das sequências de vídeo que apresentou maior redução no tempo de codificação, entretanto, ao custo de um acréscimo em BD-rate de 0,8780%. Isto indica que os modelos apresentaram um erro maior nas decisões em encerrar a ME sem executar a etapa de predição *affine* (falsos-positivos). Ainda assim, o acréscimo em BD-rate ainda é baixo quando comparado a redução em tempo de codificação que o LEAP pode oferecer. Por outro lado, é possível perceber que para as sequências de vídeo *Campfire*, *FoodMarket4*, e *Tango2*, os resultados obtidos em termos de BD-rate foram negativos. Isto indica que o processo de codificação com o LEAP foi capaz de reduzir a taxa de bits, mantendo a mesma qualidade visual que o codificador original. Este comportamento é derivado, principalmente, por dois motivos. O primeiro deles deve-se ao fato de que a baixa quantidade de quadros codificados para as sequências das classes A1 e A2 implica em um menor número de quadros de referência disponíveis. Neste sentido, há uma menor probabilidade de previsões erradas fornecerem piores referências para

quadros posteriormente codificados, que ao longo do tempo podem causar um efeito cascata indesejável, e causarem um efeito negativo na eficiência de codificação. Por outro lado, mesmo uma decisão errada na predição *affine* pode direcionar as etapas de predição unilateral e predição bilateral para outras regiões do vídeo e, eventualmente, encontrarem referências com custo inferior. Novamente, este efeito tende a se dissipar conforme os quadros foram sendo codificados e servirem como referências futuras, entretanto, para a classe de vídeo A2, o resultado obtido foi de -0,0359%, em termos de BD-rate, que caracterizam ganhos na eficiência de codificação.

Por fim, uma das preocupações em relação à implementação dos modelos de Florestas Aleatórias no codificador VVC era quanto ao custo relativo à inserção dos modelos. Neste sentido, na última coluna da Tabela 9 foi avaliado o tempo de processamento médio gasto executando os modelos treinados em relação ao tempo total de codificação. Nota-se que em nenhum caso o tempo de execução dos modelos foi superior a 0,1%, isto comprova que o custo inserido pelo LEAP, em termos de tempo de processamento, é negligenciável. Em média, o custo obtido por inserir o LEAP no codificador VVC foi de apenas 0,07% do tempo total de codificação. Além disso, é importante ressaltar que este custo é considerado nos cálculos de redução de tempo do LEAP em relação ao VTM original.

Para avaliar os ganhos obtidos na redução do tempo de codificação com a utilização do LEAP, a Tabela 10 apresenta a porcentagem de escolha nas decisões de cada modelo, onde uma decisão *Não* indica que o processo de codificação não pode ser encerrado naquele ponto, enquanto que *Sim* indica que a codificação deve ser encerrada daquele ponto em diante. Pelos resultados obtidos, é possível perceber que os modelos tem uma preferência em executar a primeira etapa de 4-parâmetros da predição *affine* para as CUs de tamanho menor. Este cenário não ocorre no modelo 3A, que comporta as CUs de tamanho 128×128, onde em 61,03% dos casos o modelo decidiu

Tabela 10 – Estatísticas de predição para cada modelo do LEAP.

Classe	Decisão	F	D	C	B	A2	A1	Média
Mod_1A	Não	65,96	57,35	59,69	64,24	50,25	60,03	**59,59**
	Sim	34,04	42,65	40,31	35,76	49,75	39,97	**40,41**
Mod_1B	Não	18,42	8,78	14,45	28,45	29,18	58,93	**26,37**
	Sim	81,58	91,22	85,55	71,55	70,82	41,07	**73,63**
Mod_2A	Não	69,75	48,86	50,36	58,60	41,25	55,32	**54,02**
	Sim	30,25	51,14	49,64	41,40	58,75	44,68	**45,98**
Mod_2B	Não	22,60	10,89	13,36	20,70	13,27	43,70	**20,75**
	Sim	77,40	89,11	86,64	79,30	86,73	56,30	**79,25**
Mod_3A	Não	65,50	29,74	29,36	35,80	25,33	48,07	**38,97**
	Sim	34,50	70,26	70,64	64,20	74,67	51,93	**61,03**
Mod_3B	Não	16,09	4,31	7,64	9,83	8,59	29,55	**12,67**
	Sim	83,91	95,69	92,36	90,17	91,41	70,45	**87,33**

em encerrar a execução. Por outro lado, os modelos apresentam uma tendência de não executar a etapa de 6-parâmetros da predição *affine* a medida que o tamanho da CU cresce. Novamente para as CUs de tamanho 128×128, em 87,33% dos casos o modelo decidiu por encerrar a execução sem processar a etapa de 6-parâmetros.

5.1 Comparação com Trabalhos Relacionados

Nesta seção é apresentado uma comparação do LEAP com outro trabalho relacionado com foco em reduzir o esforço computacional da etapa de predição *affine*. Por tratar-se de um tema recente nos tópicos de redução de complexidade em codificadores de vídeo, é importante salientar que, a escolha de apenas um trabalho para comparação deve-se ao fato de não haver outros trabalhos na literatura com foco especificamente em simplificar a etapa de predição *affine* no codificador VVC. Neste sentido, Park; Kang (2019) propõem um método de terminação precoce com base em informações de CUs vizinhas, e uma simplificação eliminando quadros de referência que apontam para direções distintas à avaliada, capaz de reduzir o esforço computacional do codificador VVC em 4%, em média.

A Tabela 11 apresenta uma comparação do LEAP com o trabalho relacionado de Park; Kang (2019). Ambos os trabalhos seguem as recomendações presentes na CTC (BOSSEN, 2020) e, portanto, utilizam as mesmas configurações de tamanho de bloco, área de busca e estruturas de particionamento. Além disso, o perfil de codifi-

Tabela 11 – Comparação com o trabalho relacionado de Park; Kang (2019).

Classe	**Video**	**RT Total (%)**		**RT Affine (%)**		**BD-rate (%)**	
		LEAP	**Park**	**LEAP**	**Park**	**LEAP**	**Park**
D	*BasketballPass*	5	2	36	27	0,19	0,08
	BlowingBubbles	8	6	50	43	0,16	0,12
	BQSquare	9	10	54	61	0,24	0,47
	RaceHorses	5	5	29	29	0,36	0,08
C	*BasketballDrill*	9	3	52	27	0,26	0,06
	PartyScene	7	4	44	40	0,21	0,26
	BQMall	8	5	47	39	0,17	0,05
	RaceHorsesC	5	2	29	25	0,25	0,06
B	*BQTerrace*	8	8	54	60	0,06	0,04
	BasketballDrive	8	5	43	34	0,16	0,08
	Cactus	7	4	43	35	0,03	0,11
	RitualDance	14	3	62	21	0,88	0,04
	MarketPlace	11	5	44	39	0,23	0,10
A2	*ParkRunning3*	6	4	29	35	0,16	0,05
	DaylightRoad2	7	6	28	41	0,04	0,14
	CatRobot1	12	7	49	45	0,18	0,05
A1	*FoodMarket4*	15	5	72	28	-0,06	0,04
	Tango2	10	7	46	37	-0,03	0,04
-	**Média**	**9**	**5**	**45**	**37**	**0,19**	**0,10**

cação escolhido foi o *Random Access*. Entretanto, em alguns aspectos os trabalhos diferem, por exemplo, o trabalho relacionado codifica os 100 primeiros quadros das sequências de vídeo de resolução UHD, enquanto que neste trabalho somente os 16 primeiros quadros das sequências de vídeo UHD foram codificados. Além disso, a versão do codificador utilizada no trabalho relacionado foi o VTM-3.0, enquanto que este trabalho utiliza o VTM-9.0, que implementa mudanças significativas nos algoritmos de predição e otimiza as etapas de busca da predição *affine*. Outra diferença presente no trabalho de Park; Kang (2019) foram as instruções SIMD habilitadas, enquanto que a metodologia seguida neste trabalho manteve essas instruções desabilitadas. Esta diferença é bastante significativa, dado que a predição *affine* se beneficia das instruções otimizadas presentes nos processadores que oferecem suporte a este tipo de instrução. Por fim, para uma comparação mais justa, somente as sequências de vídeo presentes em ambos trabalhos foram mantidas. Além disso, os valores foram arredondados para manter a mesma precisão. Os resultados comparados são em termos de redução do tempo de codificação total, redução do tempo de codificação da etapa de predição *affine*, e eficiência de codificação medida em termos de BD-rate.

Quando verificado a redução de tempo total de codificação, é possível perceber que a LEAP apresenta uma redução superior à alcançada por Park; Kang (2019) na maioria dos casos. As exceções são em *RaceHorses* e *BQTerrace*, onde houve um empate devido ao arredondamento. Além disso, em *BQSquare* o esquema proposto obtém uma redução 1% menor que o trabalho relacionado. Este é um caso específico, onde a técnica de Park; Kang (2019) parece se beneficiar, visto que é nessa sequência de vídeo que se encontra a maior redução de tempo de codificação. Quando observados os resultados médios, é possível notar que o LEAP foi capaz de reduzir 9% do tempo total de codificação, cerca de 80% maior que o resultado médio obtido pelo trabalho relacionado. No âmbito de redução do tempo de codificação da da predição *affine* um cenário similar pode ser observado, sendo que, em média, o LEAP apresenta uma redução média de 45%, cerca de 22% maior que trabalho relacionado. Quanto a eficiência de codificação, o LEAP apresentou um BD-rate médio de 0,19%, com um acréscimo de apenas 0,09% em relação à (PARK; KANG, 2019). Esta diferença deve-se principalmente a sequência de vídeo *RitualDance*, onde o acréscimo no LEAP foi de 0,84% em relação ao trabalho relacionado. Mesmo tratando-se de uma eficiência de codificação menor, ainda é um impacto insignificante no processo de codificação, dado que a redução em tempo de codificação foi bastante expressiva. Em geral, é possível verificar que o LEAP é capaz de reduzir mais o tempo de codificação do codificador VVC, cerca de 80% maior, em troca de um acréscimo insignificante em BD-rate de apenas 0,09%, quando comparado ao trabalho relacionado (PARK; KANG, 2019).

6 CONCLUSÃO

A redução do esforço computacional em codificadores atuais é um tema relevante nos tópicos de codificação de vídeo. Neste sentido, a estimação de movimento é um dos módulos de grande interesse de pesquisa, uma vez que representa uma das etapas que mais demandam tempo de processamento nos codificadores atuais.

Por outro lado, para obter uma eficiência de codificação superior ao seu antecessor, o mais novo padrão de codificação, conhecido por VVC, adicionou a sua etapa de estimação de movimento uma nova ferramenta de predição. Chamada de predição *affine*, esta ferramenta busca mapear as transformações não translacionais presentes nos vídeos digitais. Tal tarefa não é suportada nos padrões de codificação anteriores, devido ao seu alto requisito em processamento. Quando analisado, foi observado que o módulo de ME do VVC pode representar até 59,36% do tempo total de codificação, sendo, portanto, a etapa que mais demanda esforço computacional do codificador. Também foi verificado que o custo para adicionar a etapa de predição *affine* na ME do VVC é alto, sendo que esta etapa pode representar até 47% do tempo total de codificação da ME, tornando-se a etapa que mais demanda esforço da estimação de movimento do VVC.

Ainda no âmbito da predição *affine*, foi verificado que, na grande maioria dos casos, os vetores de movimento gerados a partir da execução desta etapa não são escolhidos ao final da execução da estimação de movimento, desperdiçando o processamento da etapa de predição *affine*. Em média, 61,19% das execuções das etapas de predição *affine* são desperdiçadas ao final da execução da estimação de movimento, ou seja, não são escolhidas como melhor modo de predição.

Para reduzir o esforço computacional presente na etapa de ME do codificador VVC, este trabalho apresentou um esquema baseado em aprendizado de máquina, chamado de *Learning-based Affine Prediction* (LEAP). O esquema proposto define seis modelos de Florestas Aleatórias, que tem o objetivo de decidir em encerrar a execução da etapa de predição *affine* em dois estágios: anterior a etapa de 4-parâmetros, e anterior a etapa de 6-parâmetros. Os modelos propostos apresentaram uma acurácia média de 91,33% para o conjunto de testes avaliado.

Uma vez implementados no codificador de referência do VVC, resultados experimentais mostraram que o LEAP foi capaz de reduzir o esforço computacional de todo o processo de codificação em 8,49%, em média. Além disso, o tempo de execução da etapa de predição *affine* foi reduzido em 46,94%, quando comparado ao codificador original. Para obter tal redução, houve um acréscimo no BD-rate de apenas 0,1821%, que indica uma perda insignificante na eficiência de codificação. Quando comparado com o único trabalho relacionado encontrado com foco na redução de esforço computacional na etapa de predição *affine*, o LEAP foi capaz de apresentar taxas de redução de tempo de processamento superiores, com um acréscimo insignificante em termos de BD-rate, provando-se uma solução eficiente para redução do esforço computacional do codificador VVC.

Por fim, é importante ressaltar que a predição *affine* é uma técnica independente do padrão VVC e, portanto, o esquema proposto pode ser utilizado em outras implementações que utilizam desta estrutura. Além disso, outras técnicas para redução de complexidade do codificador VVC podem ser aliadas ao LEAP para fornecerem uma redução ainda maior. Esta estratégia tem grande potencial de ofertar altas taxas de redução de esforço computacional com baixa penalidade na eficiência de codificação e, portanto, deve ser explorada em trabalhos futuros.

REFERÊNCIAS

ADHURAN, J.; FERNANDO, A.; KULUPANA, G.; BLASI, S. Affine Intra-prediction for Versatile Video Coding. In: EUROPEAN SIGNAL PROCESSING CONFERENCE (EUSIPCO), 2020., 2021. **Anais...** [S.l.: s.n.], 2021. p.545–549.

AGUI, T.; OKAWA, M.; NAKAJIMA, M. Automatic interpolation method of gray-valued images using texture mapping. **Proceedings of SPIE - The International Society for Optical Engineering**, [S.l.], v.1092, 05 1989.

ALTUNBASAK, Y.; TEKALP, A. M. Closed-form connectivity-preserving solutions for motion compensation using 2-D meshes. **IEEE Transactions on Image Processing**, [S.l.], v.6, n.9, p.1255–1269, 1997.

AMESTOY, T. et al. Random Forest Oriented Fast QTBT Frame Partitioning. In: ICASSP 2019 - 2019 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), 2019. **Anais...** [S.l.: s.n.], 2019. p.1837–1841.

AMESTOY, T. et al. Tunable VVC Frame Partitioning Based on Lightweight Machine Learning. **IEEE Transactions on Image Processing**, [S.l.], v.29, p.1313–1328, 2020.

ARMSTRONG, M. et al. High Frame-Rate Television. **SMPTE Motion Imaging Journal**, [S.l.], v.118, p.54–59, 10 2009.

BJONTEGAARD, G. **Calculation of Average PSNR Differences between RD-curves**.

BOSSEN, F. **VTM common test conditions and software reference configurations for SDR video**. Document WG 05 MPEG Joint Video Coding Team(s) with ITU-T SG 16 JVET-T2010.ed. [S.l.: s.n.], 2020.

BRANNAN, D. A.; ESPLEN, M. F.; GRAY, J. J. **Geometry**. [S.l.]: Cambridge University Press, 1999.

BREIMAN, L. Random Forests. **Machine Learning**, [S.l.], v.45, n.1, p.5–32, 2001.

CERVEIRA, A.; AGOSTINI, L.; ZATT, B.; SAMPAIO, F. Memory Profiling of H.266 Versatile Video Coding Standard. In: IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS (ICECS), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.1–4.

CHEN, J. **Algorithm description for Versatile Video Coding and Test Model 8 (VTM 8)**. JVET-Q2002.ed. Bélgica: [s.n.], 2020.

CORREA, G.; DALL'OGLIO, P.; PALOMINO, D.; AGOSTINI, L. Fast Block Size Decision for HEVC Encoders with On-the-Fly Trained Classifiers. In: EUROPEAN SIGNAL PROCESSING CONFERENCE (EUSIPCO), 2020., 2021. **Anais...** [S.l.: s.n.], 2021. p.540–544.

DAEDE, T.; NORKIN, A.; BRAILOVKKIY, I. **Video Codec Testing and Quality Measurement**. Disponível em: <<https://tools.ietf.org/html/draft-ietf-netvc-testing-07>. Acesso em Novembro de 2020.

FAN, Y. et al. A Fast QTMT Partition Decision Strategy for VVC Intra Prediction. **IEEE Access**, [S.l.], v.8, p.107900–107911, 2020.

GONçALVES, P.; MORAES, C.; PORTO, M.; CORREA, G. Complexity-Aware TZS Algorithm for Mobile Video Encoders. **Journal of Integrated Circuits and Systems**, [S.l.], v.14, n.3, 2019.

HARRINGTON, P. **Machine Learning in Action**. USA: Manning Publications Co., 2012.

Haykin, S. **Neural Networks and Learning Machines**. [S.l.: s.n.], 2008. 906p.

HOANG, D. T.; LONG, P. M.; VITTER, J. S. Efficient cost measures for motion estimation at low bit rates. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.8, n.4, p.488–500, 1998.

HUANG, C.-L.; HSU, C.-Y. A new motion compensation method for image sequence coding using hierarchical grid interpolation. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.4, n.1, p.42–52, 1994.

HUANG, H.; WOODS, J. W.; ZHAO, Y.; BAI, H. Control-Point Representation and Differential Coding Affine-Motion Compensation. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.23, n.10, p.1651–1660, Oct 2013.

ITU-T. **Subjective video quality assessment methods for multimedia applications**. P.910.

JCT-VC. **Text of ISO/IEC FDIS 23008-2 (4th edition)**. N18277.ed. [S.l.: s.n.], 2019.

KIM, J. et al. An SAD-Based Selective Bi-prediction Method for Fast Motion Estimation in High Efficiency Video Coding. **ETRI Journal**, [S.l.], v.34, n.5, p.753–758, 2012.

KIRCH, W. (Ed.). **Pearson's Correlation Coefficient**. Dordrecht: Springer Netherlands, 2008. 1090–1091p.

LEE, O.; WANG, Y. Motion-Compensated Prediction Using Nodal-Based Deformable Block Matching. **Journal of Visual Communication and Image Representation**, [S.l.], v.6, n.1, p.26–34, 1995.

LI, L. et al. An Efficient Four-Parameter Affine Motion Model for Video Coding. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.28, n.8, p.1934–1948, 2018.

LI, X.; JACKSON, J. R.; KATSAGGELOS, A. K.; MERSERAU, R. M. Multiple global affine motion model for H.264 video coding with low bit rate. In: IMAGE AND VIDEO COMMUNICATIONS AND PROCESSING 2005, 2005. **Anais...** SPIE, 2005. v.5685, p.185 – 194.

LIN, S. et al. **Affine transform prediction for next generation video coding**. ITU-T SG16 Doc. COM16–C1016.ed. [S.l.: s.n.], 2018.

MANSRI, I. et al. Comparative Evaluation of VVC, HEVC, H.264, AV1, and VP9 Encoders for Low-Delay Video Applications. In: FOURTH INTERNATIONAL CONFERENCE ON MULTIMEDIA COMPUTING, NETWORKING AND APPLICATIONS (MCNA), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.38–43.

MERCAT, A.; VIITANEN, M.; VANNE, J. UVG dataset: 50/120fps 4K sequences for video codec analysis and development. In: ACM MULTIMEDIA SYST. CONF., 2020. **Anais...** [S.l.: s.n.], 2020. p.297–302.

NAKAYA, Y.; HARASHIMA, H. Motion compensation based on spatial transformations. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.4, n.3, p.339–356, 1994.

NASIRI, F. et al. Prediction-Aware Quality Enhancement of VVC Using CNN. In: IEEE INTERNATIONAL CONFERENCE ON VISUAL COMMUNICATIONS AND IMAGE PROCESSING (VCIP), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.310–313.

PAKDAMAN, F.; ADELIMANESH, M. A.; GABBOUJ, M.; HASHEMI, M. R. Complexity Analysis Of Next-Generation VVC Encoding And Decoding. **2020 IEEE International Conference on Image Processing (ICIP)**, [S.l.], Oct 2020.

PARK, S.; KANG, J. Fast Affine Motion Estimation for Versatile Video Coding (VVC) Encoding. **IEEE Access**, [S.l.], v.7, p.158075–158084, 2019.

PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, [S.l.], v.12, p.2825–2830, 2011.

RICHARDSON, I. E. **Video Coding Design**: Developing Image and Video Compression Systems. New York, USA: John Wiley & Sons ltd., 2002.

RICHARDSON, I. E. **The H.264 Advanced Video Compression Standard, 2nd Edition**. New York, USA: John Wiley & Sons ltd., 2010. 346p.

SALDANHA, M. et al. An Overview of Dedicated Hardware Designs for State-of-the-Art AV1 and H.266/VVC Video Codecs. In: IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS (ICECS), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.1–4.

SALDANHA, M.; SANCHEZ, G.; MARCON, C.; AGOSTINI, L. Fast Partitioning Decision Scheme for Versatile Video Coding Intra-Frame Prediction. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.1–5.

SALOMON, D. **Data Compression**: The Complete Reference. [S.l.: s.n.], 2007.

SAMUEL, A. L. Some studies in machine learning using the game of Checkers. **IBM JOURNAL OF RESEARCH AND DEVELOPMENT**, [S.l.], p.71–105, 1959.

SANDVINE. **2020 Global Internet Phenomena Report**. Disponível em: <https://www.sandvine.com/phenomena>. Acesso em: 2020-31-01.

SEFERIDIS, V. E.; GHANBARI, M. General approach to block-matching motion estimation. **Optical Engineering**, [S.l.], v.32, n.7, p.1464 – 1474, 1993.

STORCH, I. **Exploração das Distorções da Projeção ERP para Redução de Complexidade da Codificação de Vídeos Omnidirecionais**. 2020. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pelotas, Pelotas.

SULLIVAN, G. J.; BAKER, R. L. **Motion compensation for video compression using control grid interpolation**. [S.l.: s.n.], 1991. 2713-2716 vol.4p.

SZE, V.; BUDAGAVI, M.; SULLIVAN, G. J. **High Efficiency Video Coding (HEVC) Algorithms and Architectures**. [S.l.]: Springer International Publishing, 2014.

TAKAMURA, S. Versatile Video Coding: a Next-generation Video Coding Standard. **NTT Technical Review**, [S.l.], v.17, June 2019.

WIEGAND, T.; STEINBACH, E.; GIROD, B. Affine multipicture motion-compensated prediction. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.15, n.2, p.197–209, 2005.

ZHANG, N.; FAN, X.; ZHAO, D.; GAO, W. Merge Mode for Deformable Block Motion Information Derivation. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.27, n.11, p.2437–2449, 2017.

ZHANG, Q. et al. Fast CU Partition Decision Method Based on Texture Characteristics for H.266/VVC. **IEEE Access**, [S.l.], v.8, p.203516–203524, 2020.

ZHOU, Q. et al. Structure damage detection based on random forest recursive feature elimination. **Mechanical Systems and Signal Processing**, [S.l.], v.46, n.1, p.82–90, May 2014.

Apêndices

APÊNDICE A – Tempo Médio de Execução da Estimação de Movimento para cada Sequência de Vídeo

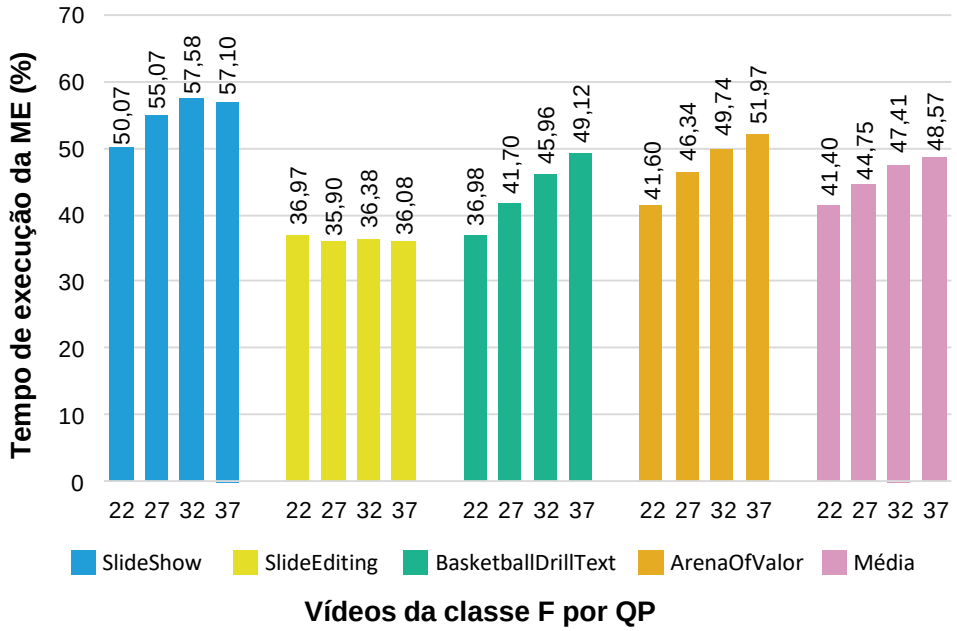


Figura 19 – Tempo de execução da ME cada sequência de vídeo da classe F.

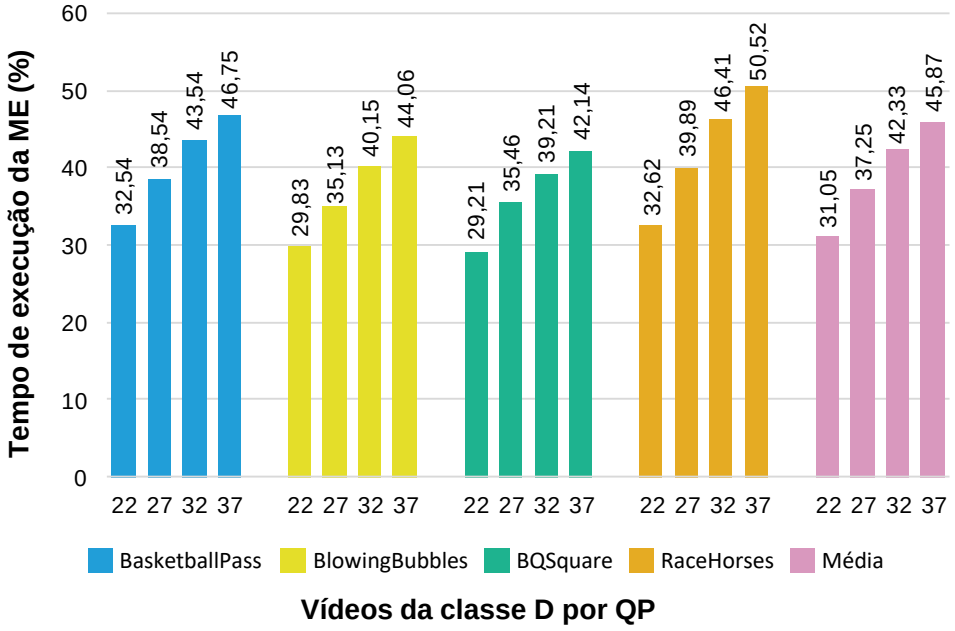


Figura 20 – Tempo de execução da ME cada sequência de vídeo da classe D.

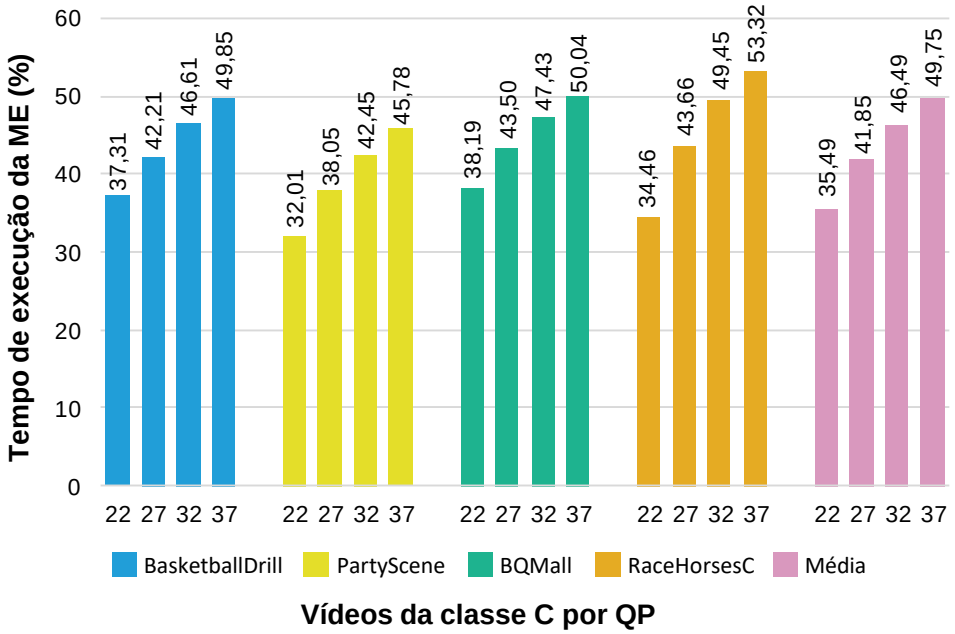


Figura 21 – Tempo de execução da ME cada sequência de vídeo da classe C.

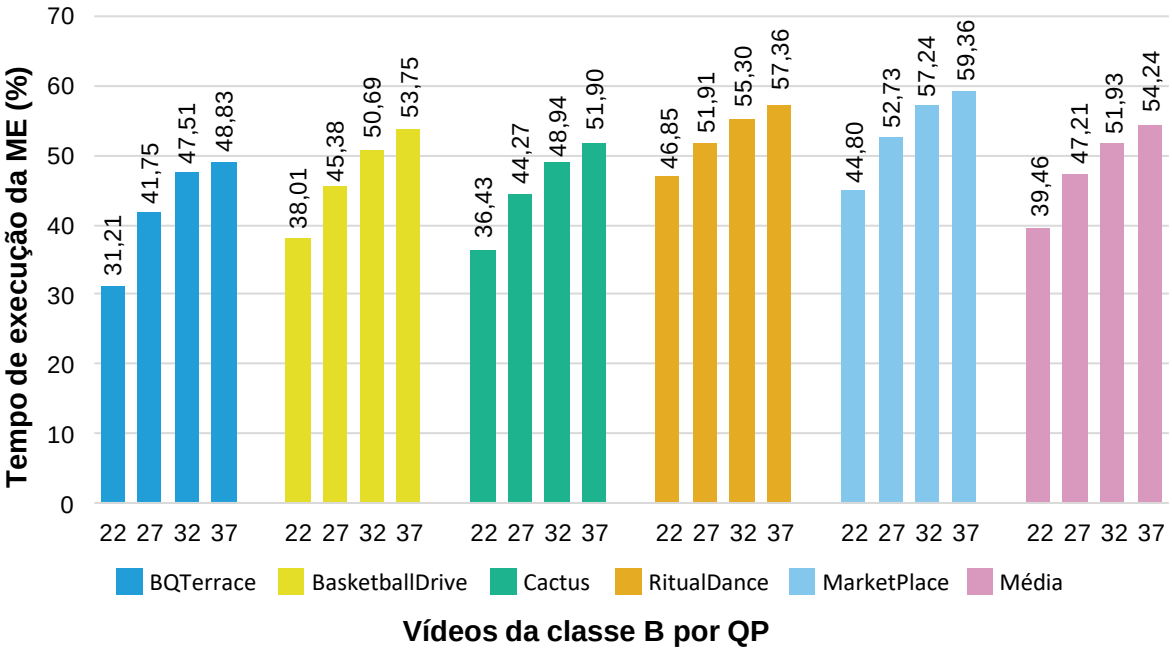


Figura 22 – Tempo de execução da ME cada sequência de vídeo da classe B.

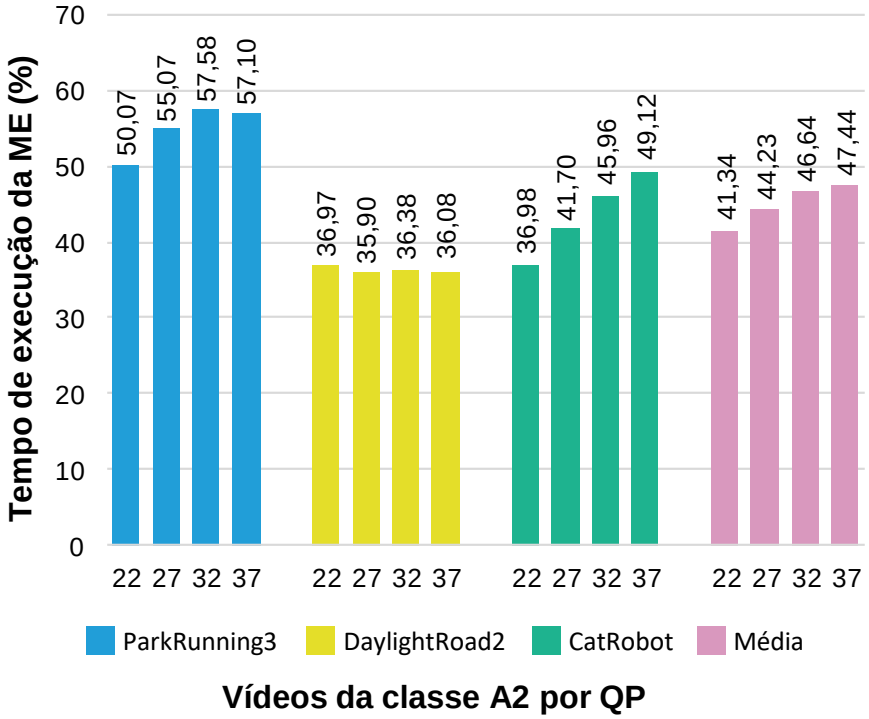


Figura 23 – Tempo de execução da ME cada sequência de vídeo da classe A2.

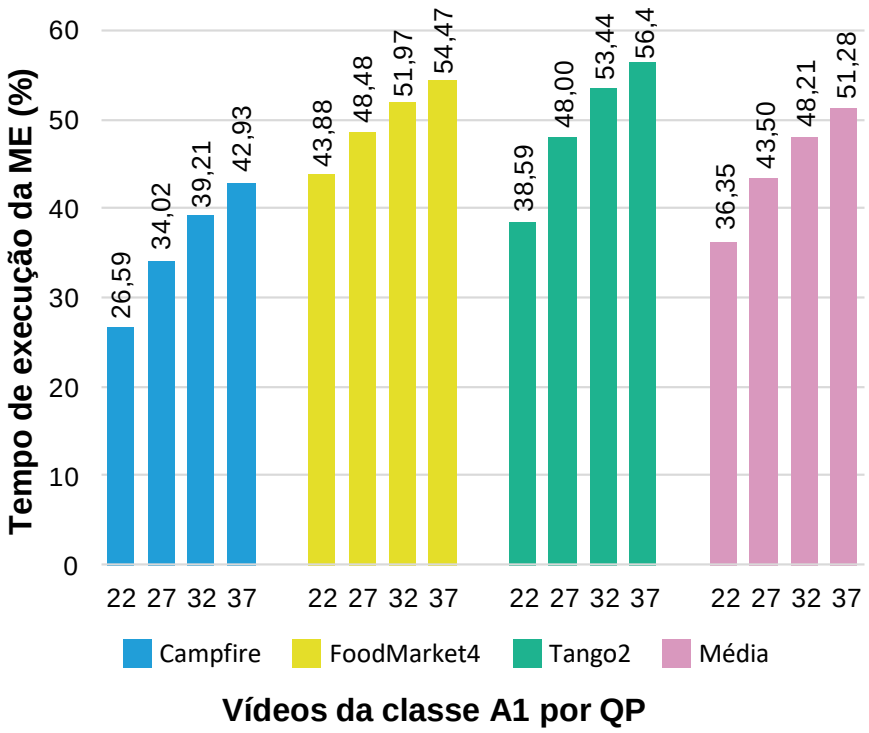


Figura 24 – Tempo de execução da ME cada sequência de vídeo da classe A1.

APÊNDICE B – Tempo de Execução de cada Etapa da Estimação de Movimento

Tabela 12 – Porcentagem do tempo de execução de cada etapa da ME para todas as sequências de vídeo analisadas.

Sequência de Vídeo	**QP**	**Predição Unilateral**	**Predição Bilateral**	**Predição Affine 4-parâmetros**	**Predição Affine 6-parâmetros**
SlideShow 1280×720	22	18,86	29,53	30,84	20,77
	27	17,25	26,42	33,88	22,45
	32	17,10	25,28	35,28	22,34
	37	18,22	24,33	36,29	21,16
SlideEditing 1280×720	22	25,48	48,61	18,47	7,44
	27	26,25	47,28	18,80	7,67
	32	26,39	46,74	19,00	7,86
	37	26,71	45,65	19,48	8,16
BasketballDrillText 832×460	22	26,22	39,30	20,79	13,70
	27	24,60	38,09	23,23	14,08
	32	23,50	37,20	25,24	14,06
	37	23,54	35,26	27,35	13,86
ArenaOfValor 1920×1080	22	24,16	36,23	23,25	16,36
	27	22,82	34,33	25,66	17,19
	32	22,26	33,30	27,08	17,36
	37	22,75	32,46	28,04	16,76
BasketballPass 416×240	22	26,91	37,79	19,77	15,53
	27	24,34	36,70	22,46	16,49
	32	23,06	36,61	24,26	16,07
	37	22,95	36,15	25,96	14,94
BlowingBubbles 416×240	22	25,79	37,76	20,84	15,60
	27	22,52	35,53	24,20	17,74
	32	20,66	34,83	26,18	18,33
	37	19,83	33,56	28,30	18,31
BQSquare 416×240	22	27,00	39,20	21,54	12,26
	27	20,96	34,11	27,29	17,65
	32	18,23	31,60	30,05	20,12
	37	17,25	28,45	32,82	21,47
RaceHorses 416×240	22	25,03	35,89	21,41	17,67
	27	22,14	34,36	24,61	18,89
	32	20,17	33,82	27,01	19,00
	37	19,83	33,30	28,81	18,06

Sequência de Vídeo	QP	Predição Unilateral	Predição Bilateral	Predição Affine 4-parâmetros	Predição Affine 6-parâmetros
BasketballDrill 832×480	22	25,80	38,75	21,31	14,14
	27	24,19	37,55	23,79	14,47
	32	23,02	36,59	25,86	14,53
	37	23,13	34,73	27,91	14,24
PartyScene 832×480	22	26,04	37,15	21,65	15,16
	27	22,67	34,79	24,97	17,57
	32	21,08	34,22	26,61	18,09
	37	20,53	32,97	28,45	18,06
BQMall 832×480	22	24,80	37,91	21,76	15,54
	27	22,90	35,13	24,89	17,08
	32	22,07	33,99	26,62	17,32
	37	22,06	32,64	28,19	17,11
RaceHorsesC 832×480	22	25,46	37,15	21,09	16,29
	27	21,65	33,84	25,98	18,53
	32	20,32	33,04	28,14	18,50
	37	20,56	31,31	30,08	18,06
BQTerrace 1920×1080	22	28,24	45,27	16,00	10,48
	27	22,16	40,47	22,75	14,62
	32	20,45	38,05	25,92	15,59
	37	20,99	35,73	28,07	15,21
BasketballDrive 1920×1080	22	25,37	41,64	19,76	13,23
	27	23,45	36,87	23,90	15,79
	32	22,72	33,65	26,73	16,90
	37	23,62	30,14	29,00	17,24
Cactus 1920×1080	22	24,84	41,70	19,46	14,00
	27	22,69	36,16	24,44	16,71
	32	21,95	33,82	26,94	17,29
	37	22,19	31,47	28,98	17,36
RitualDance 1920×1080	22	24,44	33,24	26,23	16,10
	27	23,65	30,76	28,48	17,12
	32	23,96	30,04	29,05	16,94
	37	26,13	28,63	29,49	15,75
MarketPlace 1920×1080	22	21,36	38,24	22,74	17,66
	27	19,50	31,67	27,65	21,18
	32	19,56	29,31	29,80	21,33
	37	21,16	27,60	30,94	20,29
ParkRunning3 3840×2160	22	17,96	32,48	25,47	24,09
	27	17,31	30,19	27,42	25,08
	32	17,71	31,45	27,70	23,14
	37	19,04	31,35	28,89	20,72
DaylightRoad2 3840×2160	22	19,10	38,18	23,44	19,28
	27	15,11	28,54	28,91	27,43
	32	14,58	25,41	31,37	28,64
	37	15,33	23,96	32,83	27,88
CatRobot 3840×2160	22	18,40	35,66	24,95	21,00
	27	15,81	28,82	29,31	26,06
	32	15,47	27,01	31,14	26,37
	37	16,12	26,15	32,60	25,13

Sequência de Vídeo	QP	Predição Unilateral	Predição Bilateral	Predição Affine 4-parâmetros	Predição Affine 6-parâmetros
Campfire 3840×2160	22	29,04	46,03	14,97	9,96
	27	26,16	43,75	18,44	11,64
	32	25,27	43,15	20,15	11,44
	37	26,23	38,82	23,37	11,58
FoodMarket4 3840×2160	22	21,77	39,81	25,32	13,10
	27	21,55	35,05	28,97	14,43
	32	22,20	31,65	30,66	15,49
	37	24,26	29,58	31,06	15,10
Tango2 3840×2160	22	22,08	41,65	21,32	14,95
	27	19,53	33,14	26,84	20,49
	32	19,39	29,22	29,65	21,74
	37	21,03	27,07	30,95	20,95

APÊNDICE C – Feature Importance dos Parâmetros Avaliados

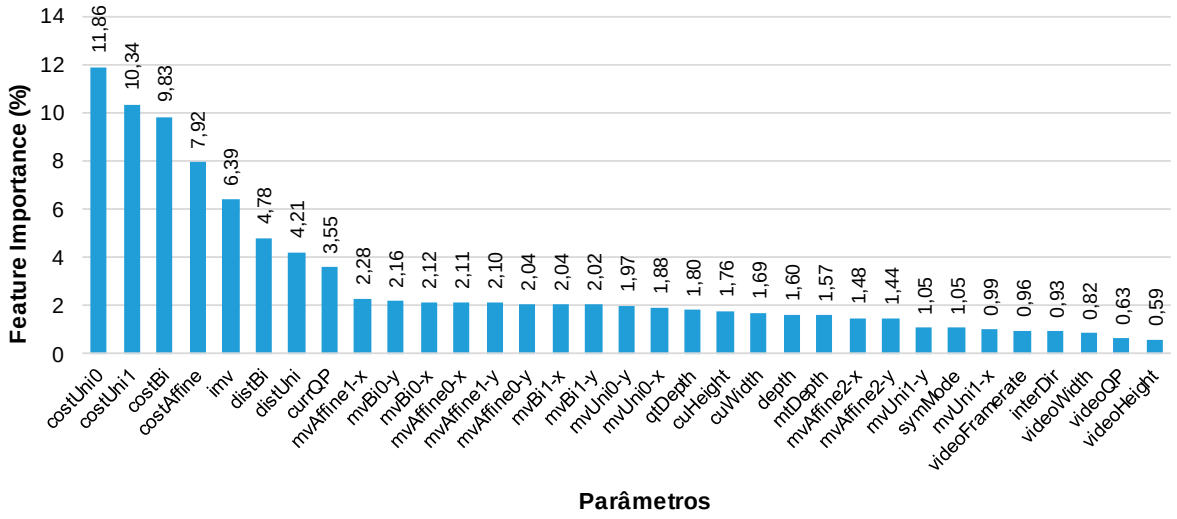


Figura 25 – Feature Importance de Todos os Parâmetros em Ordem Decrescente.

APÊNDICE D – Trabalhos publicados durante o período de Mestrado

D.1 Complexity-Aware TZS Algorithm for Mobile Video Encoders

- Autores: Paulo Gonçalves, Cândido Moraes, Marcelo Porto e Guilherme Corrêa.
- Publicado no *Journal of Integrated Circuits and Systems* em 27 de Dezembro de 2019.
- Abstract: Video applications are significantly growing in the last years, especially in embedded/mobile systems. Modern video compression algorithms and standards, such as the High Efficiency Video Coding (HEVC), achieved a high efficiency in compression ratio. However, such efficiency has caused an augment on the complexity to encode videos. This is a serious problem especially in mobile systems, which present restrictions on processing and energy consumption. This paper presents an enhanced Test Zone Search (TZS) algorithm, aiming at complexity reduction of the Motion Estimation (ME) process in the HEVC standard and focusing efficient hardware design for mobile encoders. The proposed algorithm is composed of two strategies: an early termination scheme for TZS, called e-TZS, and the Octagonal-Axis Raster Search Pattern (OARP). When combined and implemented in the HEVC reference encoder, the strategies allowed an average complexity reduction of 75.16% in TZS, with a negligible BD-rate increase of only 0.1242% in comparison to the original algorithm. Besides, the approach presents an average block matching operation reduction of 80%, allowing hardware simplification and decreasing memory access.

D.2 Learning-based Bypass Zone Search Algorithm for Fast Motion Estimation

- Autores: Paulo Gonçalves, Guilherme Corrêa, Luciano Agostini e Marcelo Porto.
- Submetido ao *Multimedia Tools and Applications* em Março de 2021.
- Abstract: Video coding has been widely explored by academia and industry in recent years, mainly due to the great popularization of video applications and multimedia-capable devices. The Motion Estimation (ME) process receives spe-

cial attention since it is one of the most complex steps in video coding. The Test Zone Search (TZS) is the main algorithm employed for integer ME in recent video codecs, such as those based on the High Efficiency Video Coding (HEVC), and has been used in the standardization process of the future Versatile Video Coding (VVC) standard. However, even though it is designed as a fast ME algorithm, the computational effort required by TZS is still very high, compromising the encoding process in multimedia-capable devices that operate on limited energy or computational resources. This work presents the Bypass Zone Search (BZS) algorithm, a learning-based solution for fast ME that improves TZS, aiming at a better trade-off between compression efficiency and computational cost. First, a set of analyses on TZS is presented, which allowed the design of two strategies to reduce the ME computational cost. The first one, named as Learning-based Bypass Motion Estimation (LBME), consists of a machine learning-based approach that predicts whether the best motion vector has already been found and bypasses the remaining ME steps. The second strategy, named as Astroid Raster Pattern (ARP), is a novel search pattern developed for the most complex TZS step, the Raster Search. By combining the two proposed strategies in BZS, the ME processing time is reduced by 60.98% (Random Access) and 63.05% (Low Delay) in comparison to TZS. The overall HEVC encoding time is reduced by 14.32% (Random Access) and 17.64% (Low Delay), with a negligible loss of 0.0837% (Random Access) and 0.04% (Low Delay) in BD-rate.