ComplexCTTP: Complexity Class Based Transcoding Time Prediction for Video Sequences Using Artificial Neural Network

Anatoliy Zabrovskiy*, Prateek Agrawal*†, Roland Mathá*, Christian Timmerer*‡, and Radu Prodan*
* *University of Klagenfurt, Klagenfurt, Austria*
† *Lovely Professional University, Punjab, India*
‡ *Bitmovin, Klagenfurt, Austria*
*Email: *anatoliy.zabrovskiy@aau.at, *†prateek.agrawal@aau.at, *roland.matha@aau.at,*
**‡christian.timmerer@aau.at, *radu.prodan@aau.at*

***Abstract*—*HTTP Adaptive Streaming* of video content is becoming an integral part of the Internet and accounts for the majority of today's traffic. Although Internet bandwidth is constantly increasing, video compression technology plays an important role and the major challenge is to select and set up multiple video codecs, each with hundreds of transcoding parameters. Additionally, the transcoding speed depends directly on the selected transcoding parameters and the infrastructure used. Predicting transcoding time for multiple transcoding parameters with different codecs and processing units is a challenging task, as it depends on many factors. This paper provides a novel and considerably fast method for transcoding time prediction using video content classification and neural network prediction. Our artificial neural network (ANN) model predicts the transcoding times of video segments for state of the art video codecs based on *transcoding parameters* and *content complexity*. We evaluated our method for two video codecs/implementations (AVC/x264 and HEVC/x265) as part of large-scale *HTTP Adaptive Streaming* services. The ANN model of our method is able to predict the transcoding time by minimizing the mean absolute error (MAE) to 1.37 and 2.67 for x264 and x265 codecs, respectively. For x264, this is an improvement of 22% compared to the state of the art.**

***Keywords*-Transcoding time prediction; adaptive streaming; video transcoding; neural networks; video encoding; video complexity class; HTTP adaptive streaming; MPEG-DASH**

I. Introduction

The demand of video applications and services is constantly increasing. Today it is a commodity to encode, distribute, share, and consume video content anywhere, anytime, and on any device [1]. Many of these services adopt a *streaming* paradigm typically deployed over the open, unmanaged Internet [2]. An important technical breakthrough and facilitator is certainly *HTTP Adaptive Streaming* (HAS). In HAS, video assets are provided in multiple versions called representations that are divided into short-term segments (*e.g.*, 2 s to 10 s) and are requested by a client device individually based on its contextual conditions (*e.g.*, network characteristics, viewing device, *etc.*) in a dynamic, adaptive manner [3, 4]. Moreover, in some implementations, such as MPEG-DASH [3], the HAS technology is independent of video compression methods and can utilize various codecs. This is especially important because we are now in a situation where we can choose from several video codecs, such as Advanced Video Coding (AVC) [5], High Efficiency Video Coding (HEVC) [6], VP9 [7], AOMedia Video 1 (AV1) [8], and Versatile Video Coding (VVC) [9]. Typically, the transcoding of video segments is a parallel process running on a high-performance infrastructure such as the cloud [10, 11]. Unfortunately, transcoding of multiple segments of a single video for adaptive streaming can take seconds or even days, depending on many technical aspects, such as video content complexity, transcoding parameters, and processing units [12]. In fact, predicting the transcoding time for the many combinations of transcoding parameters for different codecs and content types (*i.e.*, genres) is a complex task and big challenge for streaming services [13, 14].

Typically, servers and big data platforms [15] are used for distributed video transcoding without any prediction of the transcoding time. For example, Stride [15] is a distributed video transcoding system that leverages the Apache Spark [16] to speedup a video segment processing time. Adding a segment transcoding time prediction feature to Stride scheduling module can significantly improve the overall transcoding time. Cloud platforms, dedicated servers, and low-energy Internet of Things devices [11, 17] are some application domains where predicting transcoding time has a significant impact on the provisioning and scheduling of thousands of transcoding tasks [10]. Accurate prediction of transcoding times for multiple tasks allow services to avoid the load imbalance and inefficient use of resources. Moreover, modern scheduling methods [18, 19] exploit the information about task completion time to make better use of the processing infrastructure. Such information is particularly useful for those transcoding services which require multiple transcoding parameters and video content with different characteristics [14].

To decrease and improve the transcoding time prediction for videos, we propose a novel and accurate method called **ComplexCCTP**, a **Complex**ity **C**lass based **T**ranscoding

Time **P**rediction, which consist of two main phases: *(i)* data generation and *(ii)* transcoding time prediction using an artificial neural network (ANN). The main purpose of the first phase is data generation and video content complexity classification. The second phase predicts the transcoding time using the developed sequential ANN model. The ANN model is able to predict the time of video transcoding task for multiple codecs taking into account the complexity of content in the context of HAS. The main contributions of our ComplexCTTP method are summarized as follows:

- We propose a video complexity classification, with respect to the video segment's *spatial information (SI)* and *temporal information (TI)* [20, 21].
- We introduce a fast approach to measure spatial and temporal information of video segments by computing them for a transcoded segment with a low bitrate and resolution.
- The developed sequential ANN model uses the most important parameters that influence the transcoding time as input data, *i.e.*, information about the complexity of the video content, properties of the input video file, and transcoding parameters of a video codec.

To assess our method, we used a set of ten different video sequences of different types with different duration and frame rate. We evaluated our approach for the two most commonly deployed video codecs/implementations (*i.e.*, AVC/x264 and HEVC/x265) and in anticipation of the results our proposed approach achieved a mean absolute error (MAE) of 1.37 for x264 and 2.67 for x265, respectively. For x264 codec, this is an improvement of 22% compared to the state of the art results [14].

The remainder of this paper is structured as follows. Section II highlights related work. Our proposed methodology is explained in Section III. Section IV describes its implementation and evaluation results are provided in Section V. Section VI concludes the paper and highlights future work.

II. Related work

Several methods for predicting transcoding time using machine learning algorithms and neural networks for codecs x264, MPEG-4 Part 2, VP8, and H.263 are proposed in [14, 22]. The authors use the following parameters as inputs for their ANN: *bitrate, framerate, resolution, codec, number and size of I, P and B frames*. They achieved accuracy of MAE as 1.75 ± 2.834. Zakerinasab *et al.* [23] conduct an analysis of the influence of video file size on transcoding efficiency and processing time. They suggest that for faster video transcoding, the size of the video segment should be dynamically selected according to the similarity of the video frames. In their work, they make some recommendations for improving the video transcoding process, but do not predict the video processing time. Li *et al.* [24] find that the processing time of a Group of Pictures (GoP) has a good correlation with the execution time of other GOPs for the same video sequence. They propose a method for predicting video transcoding time for live streaming services based on the execution time history of GOPs. Several other works [25, 26] also present history-based predictive models. Nevertheless, due to the significant variability in the workload of continuous processing tasks, history-based models often provide low accuracy. Ma *et al.* [27] propose a video transcoding time prediction method, with respect to video segment length and targeted bitrates. The authors make predictions based on collected statistics and probabilistic theory and do not take into account the complexity of the content. Paakkonen *et al.* [28] present an architecture for predicting video transcoding metrics in a Docker-based system. The authors predict CPU utilization and transcoding time for live video transcoding tasks on virtual machine instances using machine learning models. They show that the video transcoding time for x264 codec for different instances can be predicted with average accuracy of 3-6%. Zhao *et al.* [29] proposed a model for predicting the complexity of transcoding video segments by analyzing I, P and B frames of the video sequences. The authors make predictions for MPEG-4 Part 2 and H264 codec, and use a limited number of videos and more focus on video transcoding task scheduling. Benkacem *et al.* [30] propose virtual video transcoders and approach for load balancing the transcoding tasks. The authors study transcoding behavior in different cloud environments and investigate the impact of the video duration on a video transcoder performance in terms of transcoding time. Krishnappa *et al.* [31] present several policies for online video streaming and suggest to transcode only the video bitrates that are actually requested by the client. They show that the increase in transcodig time of a single segment is almost linear with the increase in video segment duration. In turn, the authors use a limited number of transcoding parameters and presets.

Only a few studies deal with predicting video segment transcoding times for x264 but without taking into account encoding presets (*e.g.*, as known in x264, x265, VP9) or content complexity of segmented videos, such as spatial and temporal information. Similarly, limited work has been done so far for x265 and high bitrates/resolutions, which are becoming increasingly popular.

III. Methodology

This section presents our methodology comprising two phases as shown in Fig. 1: *(i) data generation* and *(ii) transcoding time prediction.*

In the data generation phase, a first step is to select videos of different complexity classes. Some sequences have minor movements, *e.g.*, moving head on a static black background *etc.*, while others have significant movements, *e.g.*, changing street view or riding jockeys, *etc.* The selected video sequences should comprise different types of

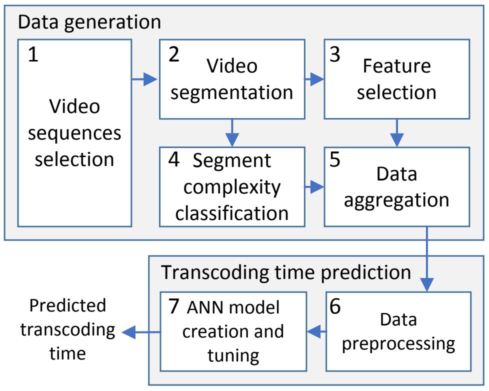
Data generation
1 Video sequences selection
2 Video segmentation
3 Feature selection
4 Segment complexity classification
5 Data aggregation
Transcoding time prediction
7 ANN model creation and tuning
6 Data preprocessing
Predicted transcoding time

Figure 1. Process flow of proposed methodology.

video content, as represented by its *spatial information* and *temporal information* [20]. The SI metric is a measure of the spatial complexity of a video content, calculated as follows:

$$SI = \max_{\forall F_n}\{\sigma\left[Sobel\left(F_n\right)\right]\}, \tag{1}$$

where F_n is a video frame at time instance n, σ is the standard deviation across all the pixels in the *Sobel* filter of its luminance component, and max selects the maximum standard deviation across all the frames in the video. The TI metric represents the amount of motion in a video sequence, defined based on a *motion difference* function M_n that represents the difference between the luminance components for identically located pixels in two consecutive frames F_n and F_{n-1}:

$$M_n(i,j) = F_n(i,j) - F_{n-1}(i,j), \tag{2}$$

where $F_n(i,j)$ is the frame pixel located at row i and column j at time instance n in the sequence. The TI metric is the maximum value of standard deviation of $M_n(i,j)$ for all the pixels:

$$TI = \max\{\sigma\left[M_n(i,j)\right]\}. \tag{3}$$

The second step is to split the video sequence into segments of, *e.g.*, 2 s or 4 s length. The segment length is one of the most important parameters in adaptive streaming, because each segment usually begins with a random access point to allow dynamic switching between representations on the client side [32].

The feature selection step plays a pivot role in prediction tasks. We extract important features for our data collection from each video segment. Each record in our dataset contains the following fields: *codec_type, segment_name, encoding_bitrate, segment_duration, width, height, encoding_preset, SI, TI, and transcoding_time.*

For the segment complexity classification, we assign a video complexity class to each video segment based on its spatial information and temporal information [21]. We define four complexity classes HH, LL, HL, LH referred to as *high TI & high SI*, *low TI & low SI*, *high TI & low SI* and *low TI & high SI*, respectively.

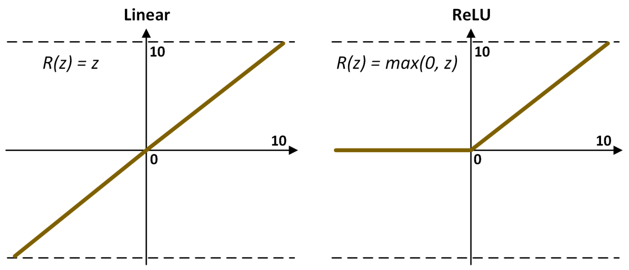

Figure 2. Linear and ReLU Activation function.

After defining the complexity class and extracting the remaining features of each video segment, we prepare the entire dataset for the next phase, *i.e.*, the *transcoding time prediction*.

Before implementing an ANN training model to predict the transcoding time of video segments, we first pre-process the generated data, *i.e.*, we reduce the number of records by calculating only the maximum and minimum transcoding time for the same combinations of transcoding parameters and *complexity class*. Then we distribute the entire dataset between training and testing dataset.

After preparing training and testing data, the next step is to select an appropriate neural network model [33]. We choose a sequential model for training because in this model only the first layer needs to receive the information about the input shape while the remaining layers do infer the input shapes automatically. Once the model is fixed based on the data, the next step is to select an appropriate activation function [34] for the neural network training. We use two types of activation functions for our model, *i.e.*, linear and non-linear. In the linear activation function, the output does not confine between any range and the function produces a line as shown in Figure 2 and, thus, we use it as output layer activation function. For hidden layers we use a rectified linear unit (ReLU) as non-linear activation function. From Figure 2, we see that the ReLU curve is half rectified. It means that for all negative input values, it turns the value into zero immediately.

To evaluate the output results of the proposed ANN model, we use MAE as shown in Eq. (4) that represents the absolute model prediction error in units of the variable, and mean squared error (MSE) as shown in Eq. (5) as the squared average difference between the actual and the predicted values.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \tag{4}$$

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2 \tag{5}$$

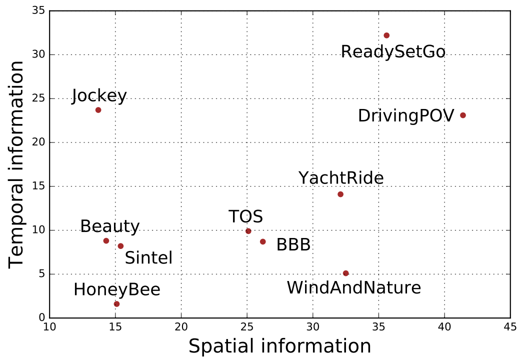

Figure 3. Average spatial information (SI) and temporal information (TI) for video sequences.

Table I
ORIGINAL VIDEO FILE CHARACTERISTICS.

Video description	Video category	Frames per second	Duration (in sec)
BBB	Animation	30	60
Beauty	Moving head	30	20
DrivingPOV	Moving cars	60	20
HoneyBee	Nature	30	20
Jockey	Sports	30	20
Sintel	Animation	24	60
TOS	Animation and real	24	60
WindAndNature	Rotating wind vanes	60	20
ReadySetGo	Sports	30	20
YachtRide	Moving yacht	30	20

Where, n is number of input instances, y_j is the target output and $\hat{y}_j$ is the actual output. Based on MAE and MSE values, we tune and update the training parameters of the ANN model. We repeat this process until we get consistent and optimal ANN performance.

IV. Implementation

This section describes the implementation of our work based on the proposed methodology.

A. Data generation

In the data generation phase, we selected ten video sequences available from a public dataset [12], which comprise different types of video content, as represented by their spatial and temporal information as depicted in Fig. 3. Thus, we can state that we used the videos that represent a wide range of possible use cases. The main characteristics of the original video sequences are given in Table I.

Using FFmpeg [35] v4.1.3, we decoded all video sequences into raw YUV format and split each sequence into segments of 2 s and 4 s duration resulting in 240 segments in total. The segment length is one of the crucial parameters in *HTTP Adaptive Streaming* because usually, each segment starts with a random access point to enable dynamic switching to other representations at segment boundaries. A segment length of 4 s shows a good trade-off with respect to streaming performance and coding efficiency [36] and is also adopted within deployments. A segment length of 2 s is also used in todays' deployments and confirms the trend towards low-latency requirements [32].

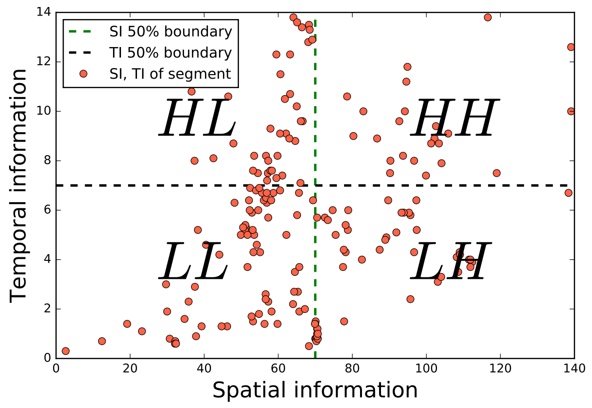

Figure 4. Video segment complexity classes (HL, HH, LH, LL).

Table II
BITRATE LADDER (BITRATE/RESOLUTION PAIRS). BITRATE VALUES ARE IN KBPS.

#	Bitrate	Resolution	#	Bitrate	Resolution
1	100	256x144	11	4300	1920x1080
2	200	320x180	12	5800	1920x1080
3	240	384x216	13	6500	2560x1440
4	375	384x216	14	7000	2560x1440
5	550	512x288	15	7500	2560x1440
6	750	640x360	16	8000	3840x2160
7	1000	768x432	17	12000	3840x2160
8	1500	1024x576	18	17000	3840x2160
9	2300	1280x720	19	20000	3840x2160
10	3000	1280x720			

After creating the YUV segments, we encoded each segment using the FFmpeg x264 library with the *veryslow* preset in order to maintain highest quality compared to the original/input video sequence. Both codecs x264 and x265 contain the same set of presets as follows: *ultrafast, superfast, veryfast, faster, fast, medium (default preset), slow, slower, veryslow, placebo*; such that, for the same video file and transcoding bitrate with slower preset, there will be the slower transcoding speed and better video quality. In our work, we did not use *placebo* as it does not give significant quality improvement compared to *veryslow* according to the official FFmpeg documentation [37]. For all segments, we calculated SI and TI metrics [21] as shown in Fig 4. Different segments that belong to the same video can have different TI and SI values. This happens because the segments of one video may contain entirely different visual complexity content. Further in this paper, we consider all these prepared segments as the source segments, and use them for multiple transcoding tasks.

We performed the transcoding on a *Intel Xeon Gold 6148* 2.4 GHz processor. Each segment was transcoded using FFmpeg's single-threaded mode which is also used in practice in a distributed cloud-based environment [11, 15]. The focus of our work is related to *HTTP Adaptive Streaming* and, thus, we adopted the *bitrate ladder* as shown in Table II which consists of a wide range of bitrates/resolutions. This selection is based on existing datasets proposed in the literature [38] and in industry best-practices and guidelines [39, 40].

We performed transcoding for both x264 and x265 on FFmpeg software and Python scripts in order to measure segment transcoding time. In total, we executed 82 080 transcoding tasks. The number of tasks were 54 720 (2 *video codecs* * 19 *bitrates* * 9 *encoding presets* * 160 *segments*) for 2 s segments and 27 360 (2 *video codecs* * 19 *bitrates* * 9 *encoding presets* * 80 *segments*) for 4 s segments. The total execution time for all transcoding tasks was approximately 580 hours. For all of the performed transcoding tasks, we formed the *Raw Transcoding Dataset* with 82 080 records containing transcoding parameters, output metrics and SI, TI information of each segment. Each record in our dataset contains the following fields: *codec_type, segment_name, encoding_bitrate, segment_duration, width, height, encoding_preset, SI, TI, and transcoding_time*.

The next step determines the video *complexity class* for each segment using SI and TI value. Calculating SI and TI metrics for the original high-resolution video segments is very time-consuming. For example, the average time to calculate the SI and TI values of a 2 s segment is about 14.2 s. Therefore, before calculating these two metrics, we quickly encoded each source video segment using the *ultrafast* preset with a low bitrate (100 kbps) and a low resolution (144p), which takes 1.07 s for 2 s segments on average. Then, we calculated SI and TI metrics with an average of 0.21 s for the same encoded video segments. By doing this, we saved almost 13 s $(14.2-(1.07+0.21))$ on an average for each segment. The optimization of determining the content complexity is also subject to future work. In this paper, the aim is to show that content complexity, such as TI and SI, can be used to better predict transcoding time.

We calculated the correlation coefficient (Pearson correlation) of TI and SI when transcoding at different quality representations according to the bitrate ladder and found a significantly strong correlation between these representations. The correlation coefficient between encoded video segments with 144p resolution and the original video segments with 2160p (or 1744p) resolution for TI and SI is 0.98 and 0.65, respectively, which presents positively strong and highly correlated relationship. Based on these findings, we decided to use the TI and SI metric values obtained from the segments transcoded with a low resolution and bitrate in our proposed work. After calculating TI and SI for segments, we defined four types of complexity classes: *(i)* Low TI - low SI (LL), *(ii)* low TI - high SI (LH), *(iii)* high TI - high SI (HH), and *(iv)* high TI - low SI (HL) as shown in Fig. 4. The dividing lines between these complexity classes were formed according to the two 50% boundaries indicated

Table III
VIDEO COMPLEXITY CLASS TYPES.

Complexity class	**TI and SI ranges**
HH (high TI, high SI)	TI (7, max], SI (70, max]
LL (low TI, low SI)	TI [min, 7), SI [min, 70)
HL (high TI, low SI)	TI [7, max], SI [min, 70]
LH (low TI, high SI)	TI [min, 7], SI [70, max]

Table IV
CHARACTERISTICS OF ANN MODEL.

Characteristic	**Optimized value**
No. of neurons in input layer	6 or 7 (with compl. class)
No. of hidden layers	3
No. of neurons in hidden layers	64/32/64
No. of neurons in output layer	2
Learning rate	default value
No. of epochs passed	500
Training data	3009
Testing data	753
Training / testing data	80%/20%
Hidden layer activation function	ReLU
Output layer activation function	Linear
Optimizer	Adadelta
Loss function	MAE
Metric	MAE, MSE
Batch size	64
Initializing weights	truncated normal

in the Table III. For example, 50% boundary of SI means that half of the segments have SI values less than 70 and half segments have values more than or equal to 70. Values of both 50% boundaries were calculated using the ceiling function [41]. Thus, for each segment, depending on its SI and TI values, one complexity class was assigned.

B. Transcoding time prediction

After identifying multiple features from each video segment, we selected the most significant parameters from the raw dataset and also derived several new parameters for the ANN model. First, we used SI and TI of each video segment to calculate a complexity class. Then we calculated $number\_of\_pixels$ parameter by multiplying $width$ and $height$. We checked redundant data entries and pre-processed the data records. For this, we reduced the number of records by only calculating the maximum and minimum transcoding time for all possible combinations of [*codec_type, complexity class, encoding_bitrate, encoding_preset, segment_duration, fps*]. Finally, we created two equal data sets for training and testing the ANN with 3762 records for each codec (x264 and x265).

To predict the transcoding time of video segments, we used the Keras Python library [42] to implement an ANN model for both codecs independently, which consist of 7 input neurons, 3 hidden layers and 2 output neurons. All input and output parameters of the ANN are schematically shown in the Fig. 5. We considered *Adadelta*, *Adagrad*, and *RMsprop* [43] optimizers for compilation of the ANN

Table V
ANN MODEL INPUT PARAMETERS.

Input parameter	Description
Encoding bitrate	video bitrates (in Kbps) from the Table II
Encoding preset	ultrafast, superfast, veryfast, faster, fast, medium, slower, slow, veryslow
Fps of input file	24, 30, 60
Complexity class	HH, LL, HL, LH
Height	144p, 180p, 216p, 288p, 360p, 432p, 576p, 720p, 1080p, 1440p, 2160p
Number of pixels	video height * video width
Segment duration	2 s and 4 s

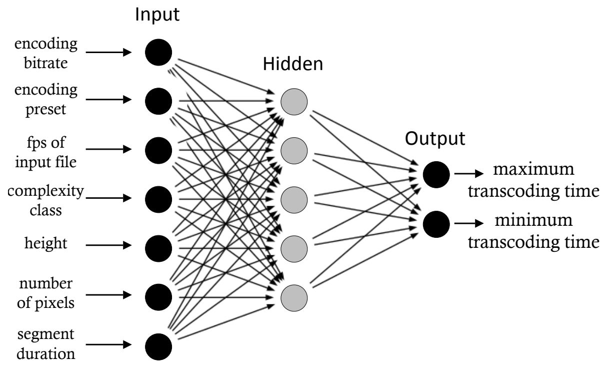

Figure 5. ANN model.

model and finally selected Adadelta after evaluating its performance with other optimizers. Adadelta is an advanced optimizer of Adagrad which adapts learning rates based on a moving window of gradient updates, rather than collecting the information of all past gradients. This special feature of Adadelta makes it better than other optimizers to learn and adjust the learning parameters by default even when multiple updates have been done. Detailed information about all the characteristics of the ANN is presented in the Table IV. We performed the ANN training of segmented video files (of 2 s and 4 s) for both codecs (x264 and x265) on the data as discussed in the previous section. All input parameters of the neural network are explained in Table V. Two ANN outputs are predicted: *maximum* and *minimum transcoding time*. To evaluate the output results of ANN model, we used MAE and MSE metrics.

V. Results and Analysis

In this section, we present the results to analyze the performance and to examine the advantages of using the proposed ComplexCTTP method for predicting transcoding times.

Typically, x265 requires more computing resources than x264, but x265 achieves higher quality than x264 with the same transcoding parameters [44]. For example, the actual transcoding time for the same 4 s segment of the *Jockey* sequence encoded with *veryslow* preset and bitrate 750 kbps (no. 6 in Table II) requires about 12 s for transcoding with x264 and 70 s with x265. For higher bitrates and resolutions, the difference is even greater, *e.g.*, transcoding with 20 Mbps bitrate (no. 19 in Table II) and *veryslow* preset for x264 and x265 takes 450 s and 680 s, respectively. Note that segment transcoding time also depends on the complexity of the content. For instance, the actual transcoding time of the 11th 4 s segment of the *Sintel* video sequence encoded with *medium* preset and bitrate 17 Mbps (no. 18 in Table II) requires about 35 s for transcoding with x264 codec. While for the next 12th segment encoded with the same transcoding parameters, the actual transcoding time is 67 s. Here, the transcoding time difference of two consecutive video segments of same video sequence is almost doubled. It happened because the content complexity of the two video segments is very different. The values of [TI and SI] for the 11th and 12th segment are [0.6 and 3.6] and [5.1 and 9.1] respectively. The 11th segment mainly contains black color and a minimum of moving objects. In order to group segments by complexity, we used four different types of complexity class as discussed in the methodology section. The average actual transcoding time for all segments for both codecs and four complexity classes, is shown in Fig. 6. There is an evident relationship between average transcoding time and complexity class. We can see that the average transcoding time is different for all complexity classes and it is shorter for the video segments with low TI and low SI value (*LL complexity class* in Table III) and vice a versa. The transcoding time increases with the complexity class of the content. If we compare the results for the *LH* and *HL* complexity classes, we can observe that the complexity class with a higher TI metric takes more transcoding time than the complexity class with high SI. This can be explained by the fact that the temporal complexity of a segment, which is based on calculating the difference in motion between the video frames of a segment, requires more time to transcode. An important implication of these findings is that each complexity class significantly

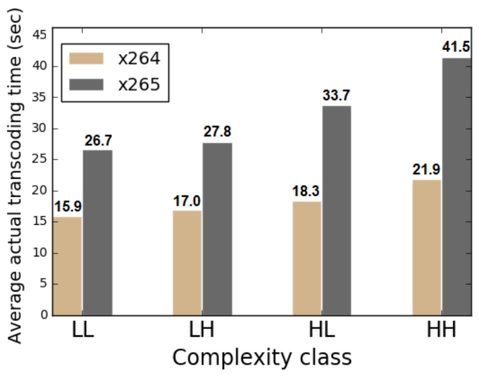

Figure 6. Average actual transcoding time for all segments belonging to a particular complexity class.

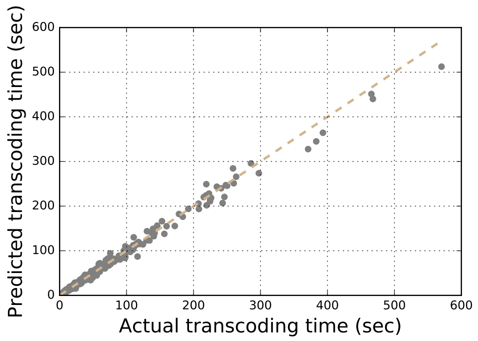

Figure 7. Predicted transcoding time vs actual transcoding time for x264 codec for testing data with complexity class.

Table VI
ANN MODEL RESULT METRICS FOR X264 AND X265 CODEC.

MAE / MSE (in sec)	x264	x265
(1) With complexity classes	1.37 / 10.96	2.67 / 45.30
(2) Without TI, SI and complexity classes	2.77 / 74.98	7.58 / 663.95
(3) With TI and SI values of 144p segments	1.48 / 13.46	3.94 / 168.36
(4) With TI and SI values of the original segments	1.32 / 10.90	2.39 / 56.01

describes the complexity of the segments in terms of the time required for transcoding. Therefore, we used the complexity class as one of the input parameters for transcoding time prediction using ANN model.

The results of MAE and MSE metric of our ANN model with complexity class input parameter compared to other developed ANN models for testing, are presented in Table VI. Based on these results, the ANN with complexity class (1) input parameter on the testing data predicts transcoding time better compared to the ANN model without any complexity class, TI and SI, (2) input parameters or ANN model with TI and SI, (3) input parameters of segments transcoded with a low resolution and bitrate. The ANN model with TI and SI input parameters of original video segments (4) has slightly better MAE for both x264 and x265 codec compared to our ANN model with complexity class (1). In turn, we can not use this ANN model (4) successfully because the calculation of TI and SI metric for the original video segments with high bitrate and resolution takes more time as mentioned in Section IV.

Prediction results using ANN produces a lower error if we use the complexity class as an (additional) input parameter. For x264, MAE and MSE values are less than for x265. We attribute this to the fact that the average transcoding time of x264 is about half of the average transcoding time of x265 (as shown in Fig. 6), and this is also reflected in the MAE metric. The predicted values of transcoding time for x264 in comparison to the actual transcoding times for testing data are shown in Fig. 7.

We compared and analysed the performance of the proposed ComplexCTTP method with results presented by Tewodros *et al.* [14]. For this, we investigated and compared the *Online Video Characteristics and Transcoding Time Dataset (OVCTT dataset)* [45] created and used by the Tewodros *et al.* with our dataset. The OVCTT dataset contains characteristics of the input and output videos and their requirements for transcoding time when transcoding video into different formats. The dataset was created based on experiments on Intel i7-3720QM CPU using FFmpeg software. The total number of transcodings for all four video codecs (x264, MPEG-4 Part 2, VP8, H.263) presented in this dataset is 68 784 (or about 17 196 per codec). While the number of transcodings for each video codec (x264 or x265) in our dataset is 41 000, which is more than double per codec compared to OVCTT dataset. A detailed comparison of both datasets in terms of transcoding time is shown in the Table. VII. The maximum values and standard deviations of transcoding times are much higher in our dataset. We did not conducted transcodings for MPEG-4 Part 2, VP8 and H.263 codec, as these are not popularly used nowadays for HAS [12]. We also compared all the transcoding parameter characteristics used in both datasets and presented them in a Table VIII. It is clearly seen that our dataset outperforms OVCTT dataset for almost all characteristics except the number of fps values. However, our dataset contains one additional value of 60 fps, which is becoming increasingly popular, especially for high resolutions and bitrates. The Tewodros *et al.* utilised only the bitrates and resolutions up to 5 Mbps and 1080p respectively as compared to our dataset where these values are up to 20 Mbps and 2160p, respectively.

Table VII
TRANSCODING TIME CHARACTERISTICS OF BOTH DATASETS.

Transcoding time characteristics	OVCTT dataset (all four codecs)	Our dataset (x264 / x265)
Number of transcodings	68784	41040 / 41400
Standard deviation	16.1	41.2 / 76.6
Minimum value (in sec)	0.18	0.59 / 0.67
Maximum value (in sec)	224.6	624.2 / 1283.5

Table VIII
TRANSCODING PARAMETERS CHARACTERISTICS OF BOTH DATASETS.

Transcoding parameter characteristic	OVCTT dataset	Our dataset
No. of presets for (x264/x265)	0 / -	9 / 9
No. of resolutions	6	11
No. of bitrates	7	19
No. of fps values	5	3
Resolution (min/max in pixels)	144p / 1080p	144p / 2160p
Bitrate (min/max in Kbps)	56 / 5000	100 / 20000
Fps value (min/max)	12 / 29.97	24 / 60

Next we developed and deployed the Python scripts on a Linux machine to calculate the input parameters for ANN models for both methods, *i.e.*, Tewodros *et al.* [14] method and our ComplexCTTP method. The Tewodros *et al.* used *bitrate, framerate, resolution, codec, number and size of I, P, B frames* as input parameters for their ANN model. The most time consuming operation in Tewodros *et al.* method

Table IX
THE AVERAGE TIME (IN SEC) REQUIRED TO CALCULATE ANN INPUT PARAMETERS FOR ONE BEAUTY VIDEO 2S SEGMENT USING BOTH THE METHODS.

Computing operations and parameters	Tewodros *et al.* method	ComplexCTTP method
I, P, B frame size and number	4.69	n/a
Transcoding to low bitrate	n/a	1.13
SI and TI metric	n/a	0.21
Total required time	**4.69**	**1.34**

Table X
COEFFICIENT OF DETERMINATION FOR BOTH THE METHODS.

Machine learning technique	Tewodros *et al.* method, R^2	ComplexCTTP method, R^2
ANN	0.958	0.994
SVR	0.942	n/a
LR	0.411	n/a

is to calculate the number and size of I, P, B frames using FFprobe [46] program, since this requires checking all the video frames of the original video file. In our ComplexCTTP method, the time-taking operation is complexity class calculation which requires to transcode original video segment using a *ultrafast* encoding preset to a low bitrate (100Kbps) and a low resolution (144p), and then calculating SI and TI complexity metrics. An example of comparison for the both methods for the *Beauty* video sequence 2 s segments of an average time required to calculate ANN input parameters is given in the Table. IX. With our ComplexCTTP method, the *percentage decrease of time* (PDT) for 2 s segments of *Beauty* video sequence is about 70%. The PDT values for all video sequences with 4 s segments are presented in Fig. 8. As we can see that for nine out of ten video sequences, the PDT values range from 53% to 80%. Only for the BBB video sequence, the PDT value is 36% (slightly lower than other video sequences). The video sequences have different PDT values because the speed of calculating the number and size of I, P, B frames by the FFprobe program in Tewodros *et al.* method directly depends on a size of segment file and a fps value. The segments of the BBB video sequence have smaller file sizes and can be processed faster. Therefore, the PDT value of the BBB video sequence is lesser.

Finally, we compared the results of predicted transcoding time of two methods. The Tewodros *et al.* applied ANN, linear regression (LR), and support vector regression (SVR) to predict the transcoding time. During training and testing of data, they achieved MAE as 1.757 ± 2.834, 1.484 ± 3.594 and 7.233 ± 9.997 for ANN, SVR, and LR respectively. With our ComplexCTTP method, we were able to minimize MAE to 1.37 for AVC/x264 which is an improvement of approximately 22% as compared to the Tewodros *et al.* method. Table X shows the coefficient of determination (predicted vs. actual) for both methods. The result shows that ComplexCTTP performs better in terms of prediction accuracy (*i.e.*, 0.958 for the Tewodros *et al.* method vs. 0.994 for our ComplexCTTP method). A comparison of our result for x265 with related work was not possible due to lack of existing methods for this codec (implementation). In turn, our ComplexCTTP method allows a much faster and more accurate prediction of the transcoding time for the x264 codec compared to the Tewodros *et al.* method.

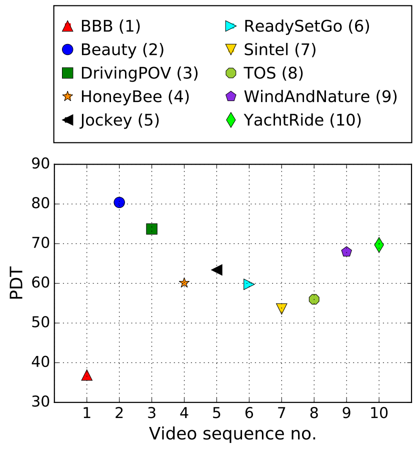

Figure 8. PDT for all ten video sequences with 4s segments.

VI. CONCLUSIONS AND FUTURE WORK

In this research, we propose the ComplexCTTP method, a novel method for fast and more accurate prediction of video transcoding time with ANN model. We developed an ANN model based on a dataset produced with x264 and x265. The ComplexCTTP method also supports an approach for assigning a video complexity class to segments. Our results show that using the video *complexity class* as an ANN input parameter yields a better prediction of the transcoding time. In particular, the results show that our created ANN model with video complexity class as input parameter for x264 and x265 is able to predict transcoding time by minimizing MAE up to 1.37 and 2.67 for x264 and x265, respectively.

Future work includes experiments on new emerging codecs and adding more sophisticated approaches for fast transcoding time prediction, such as intelligently selecting and analyzing the content complexity of a few segments of a video to make prediction about the transcoding time of the entire video. Additionally, we will investigate using the predicted transcoding time for the actual scheduling of video transcoding tasks with multiple codecs in a heterogeneous cloud infrastructure.

ACKNOWLEDGMENTS

This work has been supported in part by the Austrian Research Promotion Agency (FFG) under the *APOLLO* project and by the European Union Horizon 2020 Research and Innovation Programme under the *ARTICONF* Project with grant agreement number 644179.

References

[1] "Cisco annual internet report (2018–2023)," Cisco, 170 West Tasman Dr. San Jose, CA 95134 USA, Tech. Rep., 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] A. Javadtalab, M. Semsarzadeh, A. Khanchi, S. Shirmohammadi, and A. Yassine, "Continuous one-way detection of available bandwidth changes for video streaming over best-effort networks," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 1, pp. 190–203, 2015.

[3] I. Sodagar, "The MPEG-DASH Standard for Multimedia Streaming Over the Internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, Oct. 2011. [Online]. Available: http://dx.doi.org/10.1109/MMUL.2011.71

[4] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 562–585, Firstquarter 2019.

[5] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[6] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[7] D. Mukherjee, J. Han, J. Bankoski, R. Bultje, A. Grange, J. Koleszar, P. Wilkins, and Y. Xu, "A Technical Overview of VP9: The Latest Open-Source Video Codec," in *SMPTE 2013 Annual Technical Conference Exhibition*, Oct 2013, pp. 1–17.

[8] Y. Chen *et al.*, "An Overview of Core Coding Tools in the AV1 Video Codec," in *2018 Picture Coding Symposium (PCS)*, June 2018, pp. 41–45.

[9] B. Bross, J. Chen, and S. Liu, "Versatile Video Coding (Draft 7)," JVET-P2001, Geneva, CH, Document, Oct. 2019.

[10] M. G. Koziri, P. K. Papadopoulos, N. Tziritas, T. Loukopoulos, S. U. Khan, and A. Y. ZoJulya, "Efficient Cloud Provisioning for Video Transcoding: Review, Open Challenges and Future Opportunities," *IEEE Internet Computing*, vol. 22, no. 5, pp. 46–55, Sep. 2018.

[11] D. Vatolin, D. Kulikov, E. Sklyarov, S. Zvezdakov, and A. Antsiferova, "Video Transcoding Clouds Comparison 2019," Moscow State University, Tech. Rep., 11 2019.

[12] A. Zabrovskiy, C. Feldmann, and C. Timmerer, "Multi-Codec DASH Dataset," in *Proceedings of the 9th ACM Multimedia Systems Conference*, ser. MMSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 438–443. [Online]. Available: https://doi.org/10.1145/3204949.3208140

[13] X. Li, M. A. Salehi, Y. Joshi, M. K. Darwich, B. Landreneau, and M. Bayoumi, "Performance Analysis and Modeling of Video Transcoding Using Heterogeneous Cloud Services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 910–922, April 2019.

[14] D. Tewodros, "Proactive Management of Video Transcoding Services," PhD dissertation, Abo Akademi University, 2017.

[15] S. Sameti, M. Wang, and D. Krishnamurthy, "Stride: Distributed Video Transcoding in Spark," in *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, 2018, pp. 1–8.

[16] "Apache Spark," https://spark.apache.org, 2020, [Online; accessed 25-July-2020].

[17] O. Barais, J. Bourcier, Y. Bromberg, and C. Dion, "Towards microservices architecture to transcode videos in the large at low costs," in *2016 International Conference on Telecommunications and Multimedia (TEMU)*, July 2016, pp. 1–6.

[18] J. K. Konjaang, J. Y. Maipan-uku, and K. K. Kubuga, "An Efficient Max-Min Resource Allocator and Task Scheduling Algorithm in Cloud Computing Environment," *CoRR*, vol. abs/1611.08864, 2016. [Online]. Available: http://arxiv.org/abs/1611.08864

[19] Y. Mao, X. C. Chen, and X. Li, "Max–Min Task Scheduling Algorithm for Load Balance in Cloud Computing," in *International Conference on Computer Science and Information Technology*, 2014, pp. 457–465.

[20] ITU-T, "Subjective video quality assessment methods for multimedia applications," International Telecommunication Union, Geneva, Recommendation ITU-T P.910, Apr. 2008.

[21] P. Lebreton, W. Robitza, and S. Göring, "Command-line tool for calculating SI and TI according to ITU-T P.910," https://github.com/Telecommunication-Telemedia-Assessment/SITI, 2019, [Online; accessed 25-July-2020].

[22] T. Deneke, S. Lafond, and J. Lilius, "Analysis and transcoding time prediction of online videos," in *2015 IEEE International Symposium on Multimedia*, Dec 2015, pp. 319–322.

[23] M. R. Zakerinasab and M. Wang, "Does chunk size matter in distributed video transcoding?" in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, June 2015, pp. 69–70.

[24] X. Li, M. A. Salehi, and M. Bayoumi, "VLSC: Video Live Streaming Using Cloud Services," in *2016 IEEE International Conferences on Big Data and Cloud Computing*, Oct 2016, pp. 595–600.

[25] Jiani Guo and L. N. Bhuyan, "Load balancing in a cluster-based web server for multimedia applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1321–1334, 2006.

[26] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius, "Prediction-based dynamic resource allocation for video transcoding in cloud computing," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013, pp. 254–261.

[27] H. Ma, B. Seo, and R. Zimmermann, "Dynamic Scheduling on Video Transcoding for MPEG DASH in the Cloud Environment," in *Proceedings of the 5th ACM Multimedia Systems Conference*, ser. MMSys '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 283–294. [Online]. Available: https://doi.org/10.1145/2557642.2557656

[28] P. Pääkkönen, H. Antti, and A. Tommi, "Online architecture for predicting live video transcoding resources," *J. Cloud Comput.*, vol. 8, no. 1, pp. 132:1–132:24, Dec. 2019. [Online]. Available: https://doi.org/10.1186/s13677-019-0132-0

[29] H. Zhao, Q. Zheng, W. Zhang, and J. Wang, "Prediction-based and locality-aware task scheduling for parallelizing video transcoding over heterogeneous mapreduce cluster," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 4, pp. 1009–1020, April 2018.

[30] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, "Per-

formance benchmark of transcoding as a virtual network function in CDN as a service slicing," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, 2018, pp. 1–6.

[31] D. K. Krishnappa, M. Zink, and R. K. Sitaraman, "Optimizing the video transcoding workflow in content delivery networks," in *Proceedings of the 6th ACM Multimedia Systems Conference*, ser. MMSys '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 37–48. [Online]. Available: https://doi.org/10.1145/2713168.2713175

[32] N. Bouzakaria, C. Concolato, and J. Le Feuvre, "Overhead and performance of low latency live streaming using MPEG-DASH," in *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, July 2014, pp. 92–97.

[33] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad", "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, Nov 2018. [Online]. Available: https://doi.org/10.1016/j.heliyon.2018.e00938

[34] C. Nwankpa, W. Ijomah, and A. G. S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *ArXiv*, vol. abs/1811.03378, 2018.

[35] "FFprobe Documentation," https://ffmpeg.org/ffprobe.html, 2020, [Online; accessed 25-July-2020].

[36] S. Lederer, C. Müller, and C. Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: ACM, 2012, pp. 89–94. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155570

[37] "H.264 Video Encoding Guide," https://trac.ffmpeg.org/wiki/Encode/H.264, 2020, [Online; accessed 10-September-2020].

[38] J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Datasets for AVC (H.264) and HEVC (H.265) Evaluation of Dynamic Adaptive Streaming over HTTP (DASH)," in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys '16. New York, NY, USA: ACM, 2016, pp. 51:1–51:6. [Online]. Available: http://doi.acm.org/10.1145/2910017.2910625

[39] Twitch, "Broadcasting Guidelines," https://stream.twitch.tv/encoding/, 2019, [Online; accessed 25-July-2020].

[40] YouTube, "Live encoder settings, bitrates, and resolutions," https://support.google.com/youtube/answer/2853702, 2019, [Online; accessed 25-July-2020].

[41] "Python mathematical functions," https://docs.python.org/2/library/math.html, 2019, [Online; accessed 25-July-2020].

[42] "Keras: The Python Deep Learning library," https://keras.io/, 2019, [Online; accessed 25-July-2020].

[43] M. D. Zeiler, "Adadelta: An adaptive learningrate method," 2012, computing Research Repository (CoRR).

[44] Q. Huangyuan, L. Song, Z. Luo, X. Wang, and Y. Zhao, "Performance evaluation of H.265/MPEG-HEVC encoders for 4K video sequences," in *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*, 2014, pp. 1–8.

[45] T. Deneke, "Online Video Characteristics and Transcoding Time Dataset," https://archive.ics.uci.edu/ml/datasets/Online+Video+Characteristics+and+Transcoding+Time+Dataset, 2015, [Online; accessed 25-July-2020].

[46] "FFmpeg," https://www.ffmpeg.org/, 2020, [Online; accessed 25-July-2020].