BENCHMARKING OF VIDEO CONVERSION ON HETEROGENEOUS VIRTUAL MACHINES

Senior Project Report

In this paper, I describe and analyze benchmarking experiments involving video transcoding on heterogeneous virtual machines using cloud services. I provide all my graphs for each conversion combination in order to provide a thorough reference and comparison between cloud services, machine types, and changes in codec, framerate, or resolution.

Diana Nguyen C00232245
CMPS 490

Table of Contents

I. Introduction

Beginning in January 2020, I was introduced to the High-Performance Cloud Computing (HPCC) Lab at the University of Louisiana at Lafayette. I learned about virtual machines (VMs) and applications in cloud services and was able to further learn through research projects. During my term, I had the opportunity to be involved in two projects: Fire Detection Application and Video Streaming.

I conducted benchmarking experiments on various VMs on Chameleon Cloud (CC) and Amazon Web Services (AWS). For CC, we utilize Kernel-based VMs (KVM), which is a virtualization module in the Linux kernel that allows the kernel to function as a hypervisor. For AWS, we use the Elastic Compute Cloud (EC2) service that offers different types of VMs. For example, some instances can be General Purpose to support a wide range of workloads, Compute Optimized to support compute-bound applications, or Memory Optimized to support workloads that process large data sets in memory, etc.

With these heterogeneous VM types and features, I benchmarked a dataset of videos for each project to determine which factors may be important to utilize cloud resources efficiently. The benchmarking for the Fire Detection Application is complete; on the other hand, I was unable to complete the benchmarking for the Video Streaming project due to time limitations. Even though it is incomplete, I believe I have collected some notable data in the understanding of time-efficiency in VMs overall.

II. Fire Detection Application

A. Project

- Description
  - In the Oil and Gas industry, cloud-based Deep Neural Network (DNN) applications are becoming prevalent due to latency-sensitive inference. These applications using heterogenous cloud resources have to be time-critical in the event of disasters. Fire Detection is one of these DNN applications and utilizes machine learning. In this project, I benchmarked a dataset of videos to determine the most time-efficient detection and machine type.
- Dataset
  - The dataset used in this project includes 30 videos that are split into multiple two-second segments. The dataset can be found here: https://drive.google.com/drive/folders/1AC2Qhs_ZoRQgeKuBpo2xjoJndvPB4HMW?usp=sharing
- Video Combinations (8 cases)

Resolution	Framerate
15	240
15	480
15	720
15	1080

Resolution	Framerate
30	240
30	480
30	720
30	1080

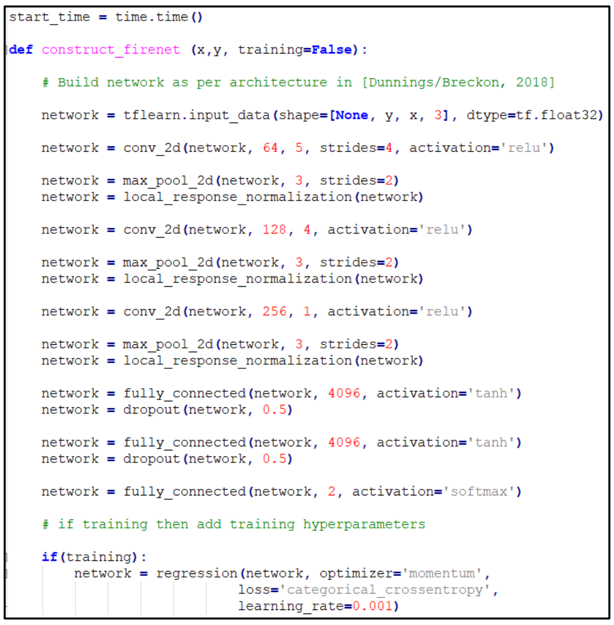
- Machine Types
  - The Fire Detection app was benchmarked on several VMs on CC and AWS, which is listed in the tables below. I will use the abbreviations I provided to refer to these types throughout the paper.
  - Note: I abbreviated the *m5a.2xlarge* as "AWS-GPA" because the "A" stands for AMD, which is the type of CPU it utilizes. Moreover, I use that abbreviation for consistency since I benchmarked on *m5.2xlarge* (AWS-GPI) in the Video Streaming project because it uses an Intel CPU.

Chameleon Cloud				
Type	Abbreviation	vCPUs	RAM (GB)	Disk (GB)
medium	CC-Medium	2	4	40
large	CC-Large	4	8	40
xlarge	CC-XLarge	8	16	40
xxlarge	CC-2xLarge	16	32	40

Amazon Web Services					
Type	Family	Abbreviation	vCPUs	Memory (GiB)	Instance Storage
c5.2xlarge	Compute optimized	AWS-CO	8	16	EBS only
g4dn.xlarge	GPU instance	AWS-GPU	4	16	1 x 125 (SSD)
inf1.xlarge	Machine Learning	AWS-ML	4	8	EBS only
m5a.2xlarge	General purpose	AWS-GPA	8 (AMD)	32	EBS only
r5.2xlarge	Memory optimized	AWS-MO	8	64	EBS only

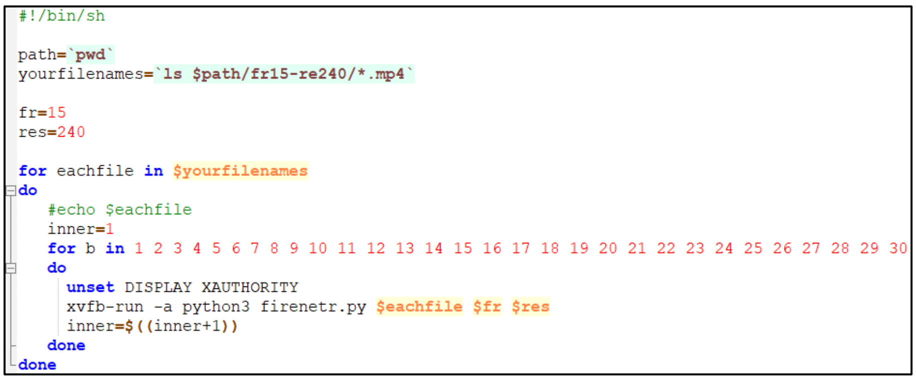
B. Method

- Scripts
  - I was provided the *FireNet* program created by Andrew Dunning and Toby Breckon at Durham University in the United Kingdom that uses real-time fire detection in video imagery using DNN. It can be referenced here: https://github.com/tobybreckon/fire-detection-cnn.
  - The program uses TensorFlow, which is an open-source platform for machine learning. Below is a sample of the *FireNet* script:



```
start_time = time.time()

def construct_firenet (x,y, training=False):

    # Build network as per architecture in [Dunnings/Breckon, 2018]

    network = tflearn.input_data(shape=[None, y, x, 3], dtype=tf.float32)

    network = conv_2d(network, 64, 5, strides=4, activation='relu')

    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)

    network = conv_2d(network, 128, 4, activation='relu')

    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)

    network = conv_2d(network, 256, 1, activation='relu')

    network = max_pool_2d(network, 3, strides=2)
    network = local_response_normalization(network)

    network = fully_connected(network, 4096, activation='tanh')
    network = dropout(network, 0.5)

    network = fully_connected(network, 4096, activation='tanh')
    network = dropout(network, 0.5)

    network = fully_connected(network, 2, activation='softmax')

    # if training then add training hyperparameters

    if(training):
        network = regression(network, optimizer='momentum',
                             loss='categorical_crossentropy',
                             learning_rate=0.001)
```

  - I was also provided with the *runFire.sh* script to run the *FireNet* program to use for each video in the dataset. Below is a screenshot of the script:

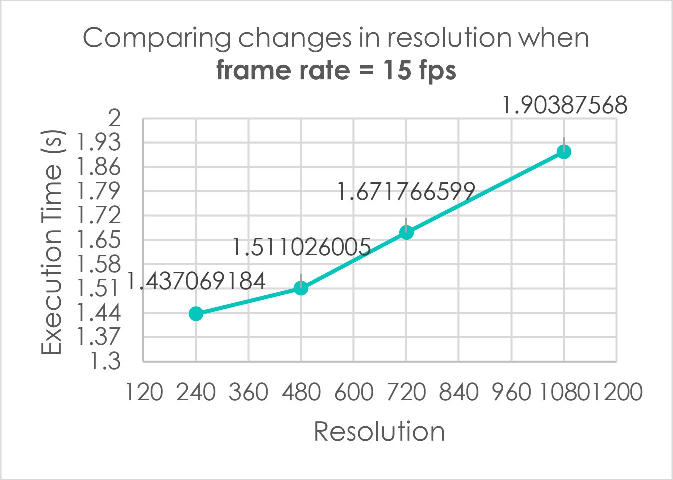


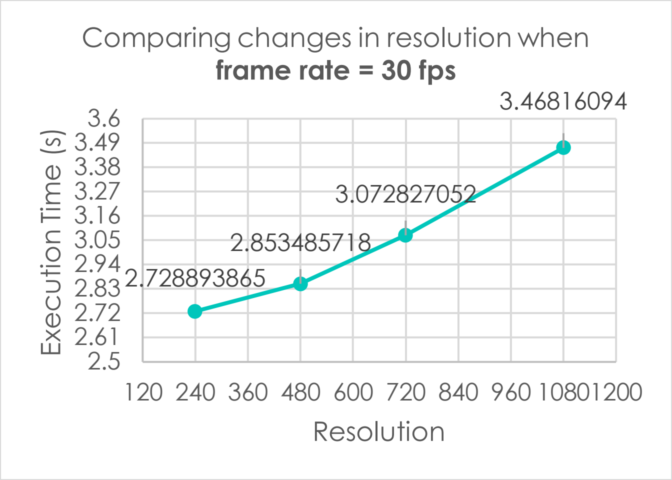
```
#!/bin/sh

path=`pwd`
yourfilenames=`ls $path/fr15-re240/*.mp4`

fr=15
res=240

for eachfile in $yourfilenames
do
    #echo $eachfile
    inner=1
    for b in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
    do
      unset DISPLAY XAUTHORITY
      xvfb-run -a python3 firenetr.py $eachfile $fr $res
      inner=$((inner+1))
    done
done
```

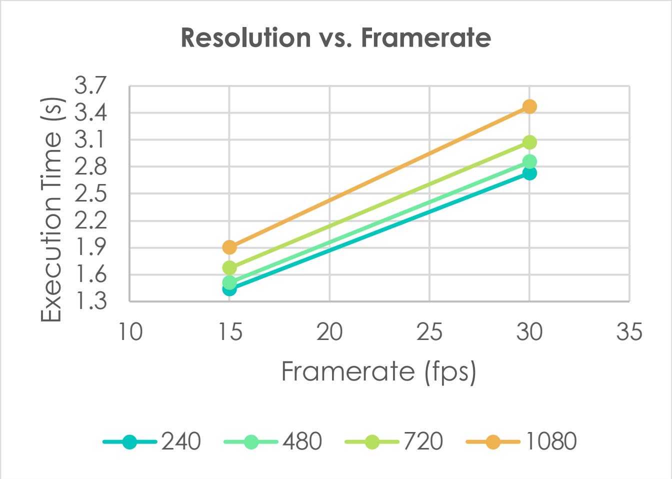
- <u>Experiment Procedure</u>
  - I used the *runFire.sh* script on each of the 9 VMs to obtain inference time (real, system, user times) for the 8 conversion cases for a total of 30 runs.
  - The data is exported to an Excel file where I analyze the output.
- <u>Analysis Procedure</u>
  - I obtained the average of the inference time for each combination in each VM. I compare the changes in resolution with the framerate as the constant as well as an overall comparison between changes in resolution and framerate.

C. Experiment Results

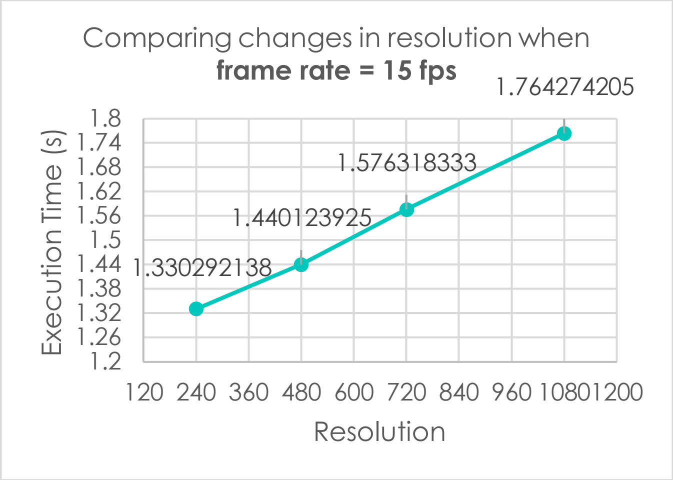
CC-Medium

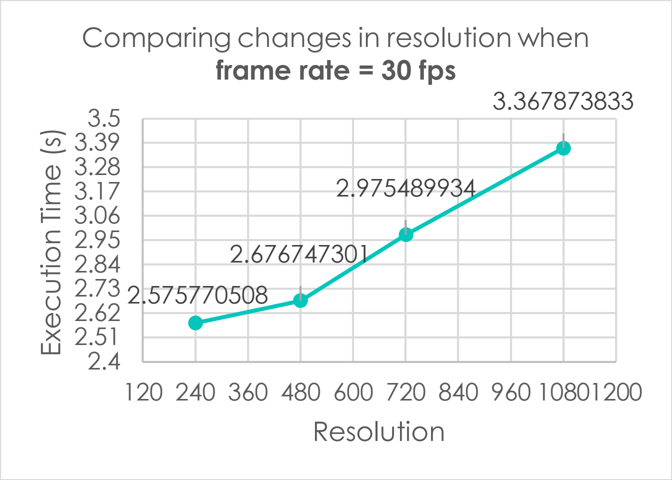


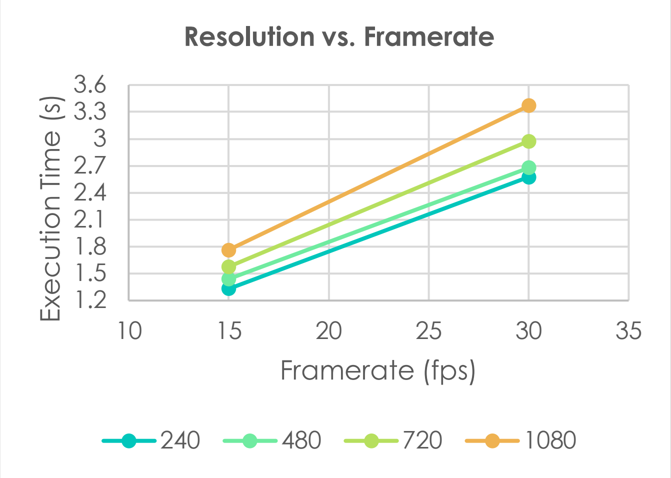




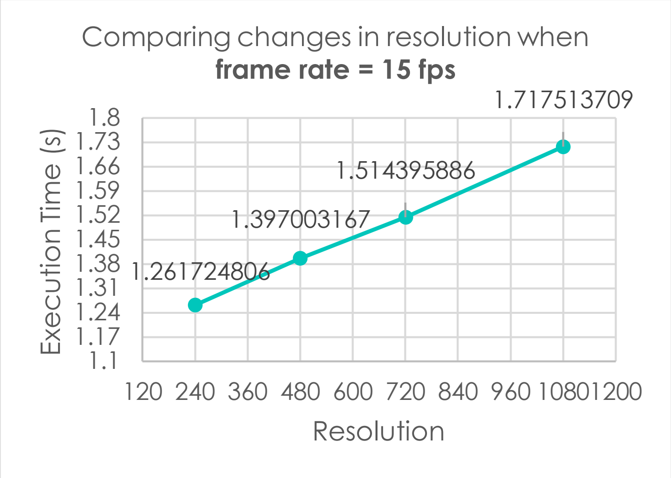
CC-Large

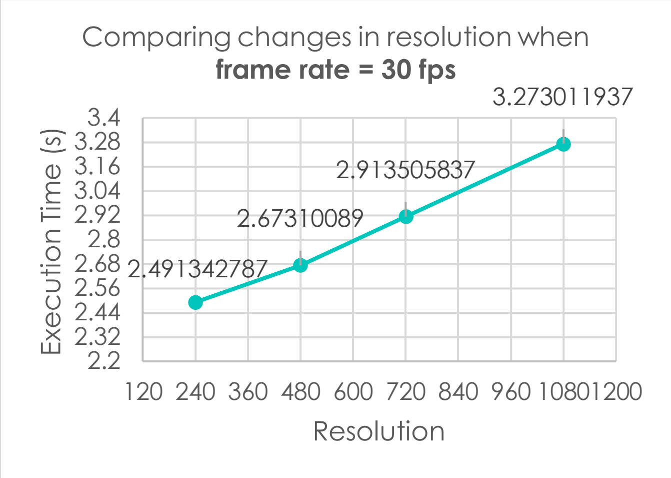


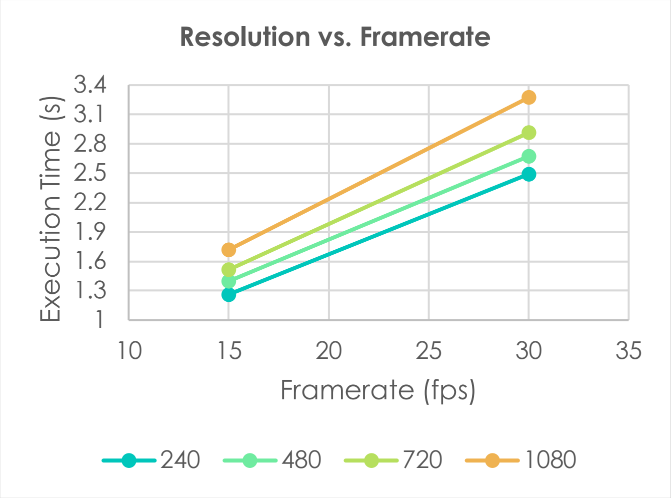




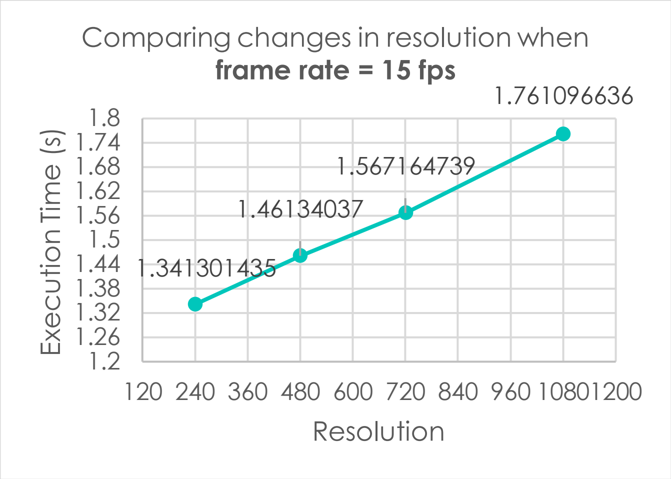
CC-XLarge

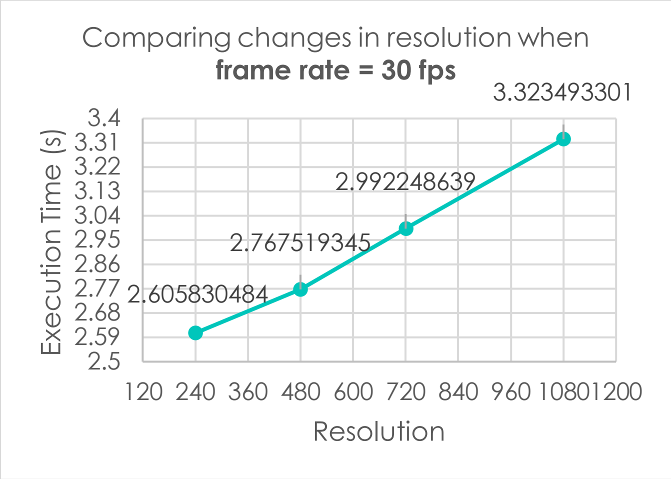


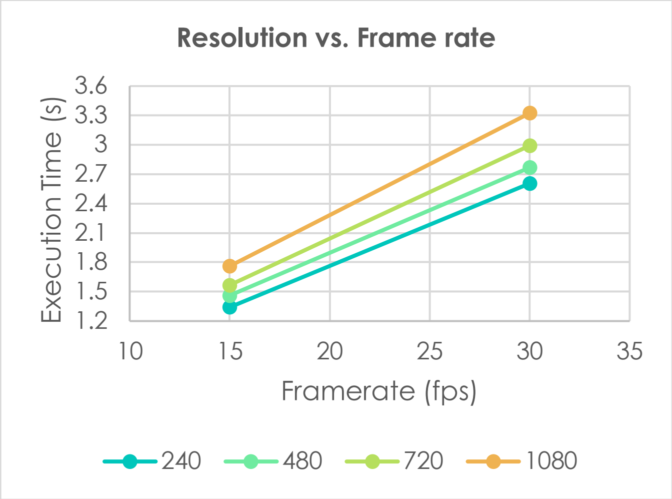




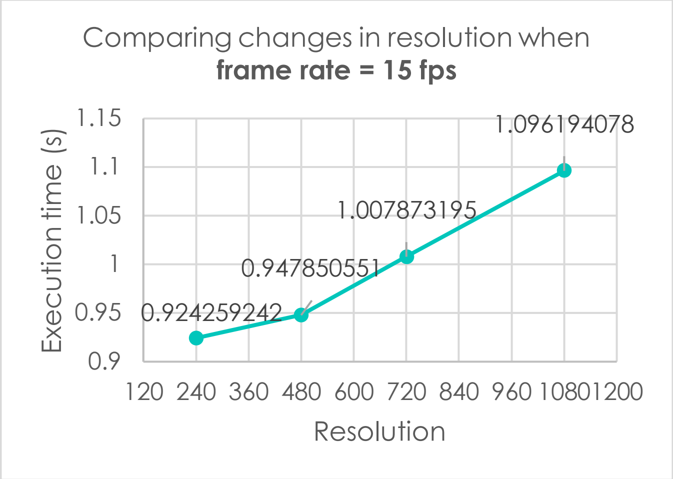
CC-2XLarge

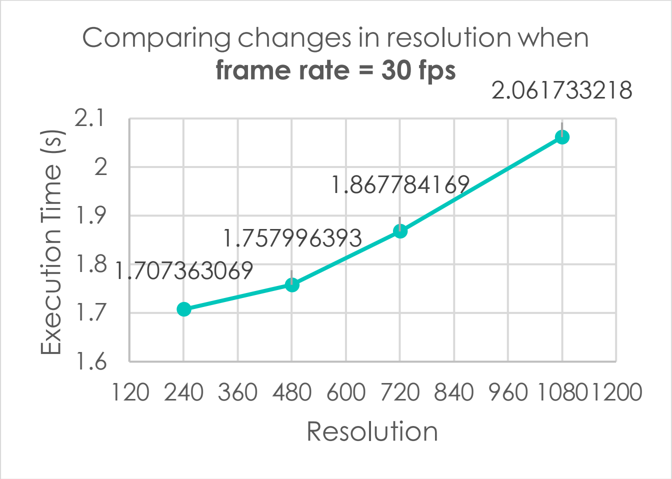


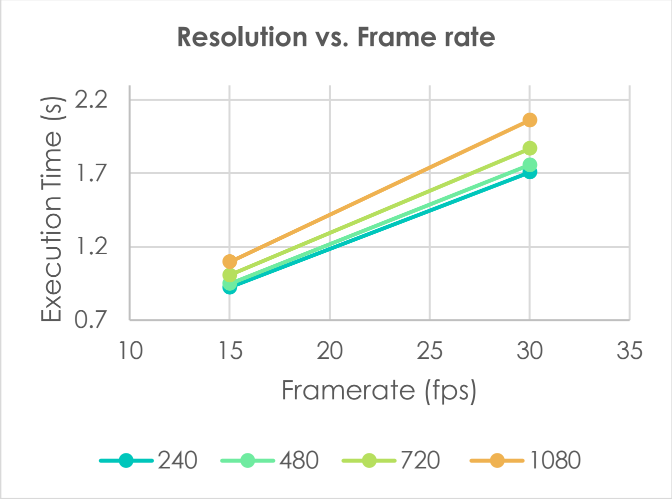



Resolution vs. Frame rate

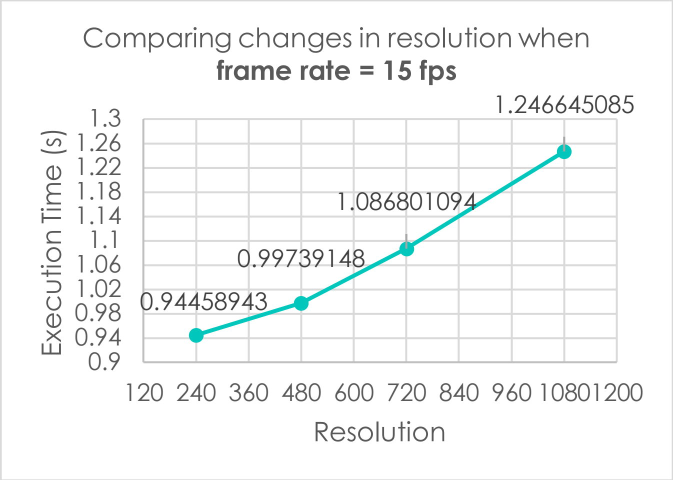
AWS-CO

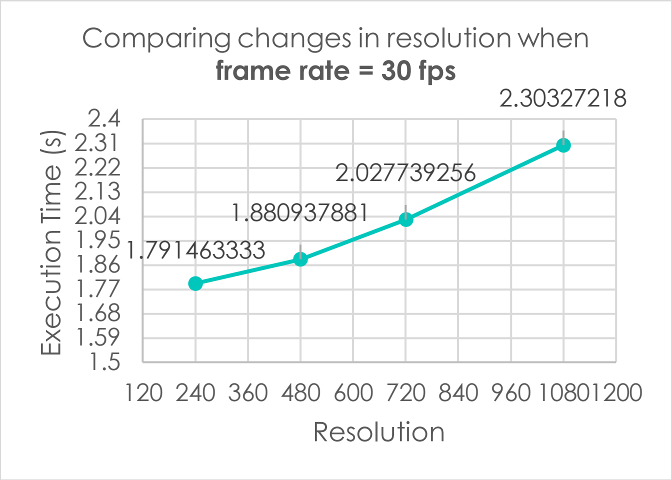

Comparing changes in resolution when **frame rate = 15 fps**

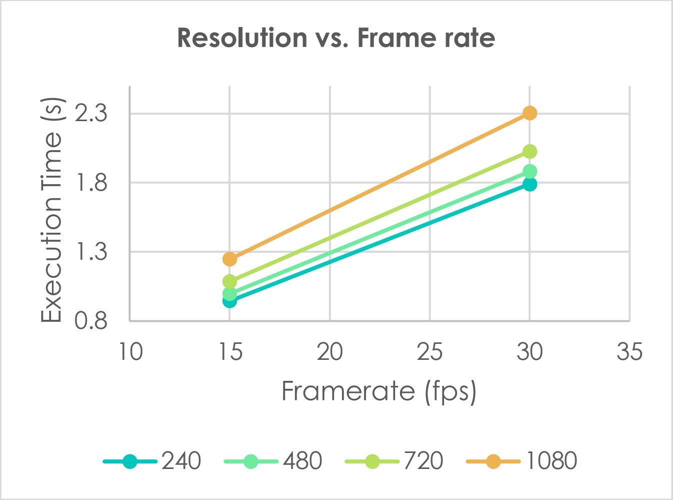

Comparing changes in resolution when **frame rate = 30 fps**


Resolution vs. Frame rate

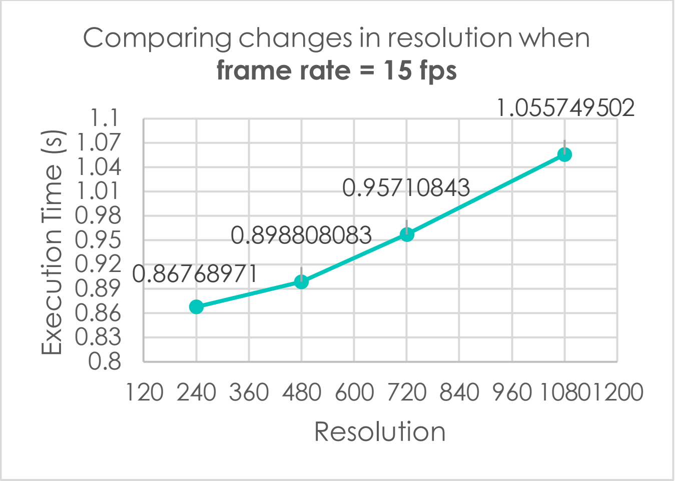
AWS-GPA

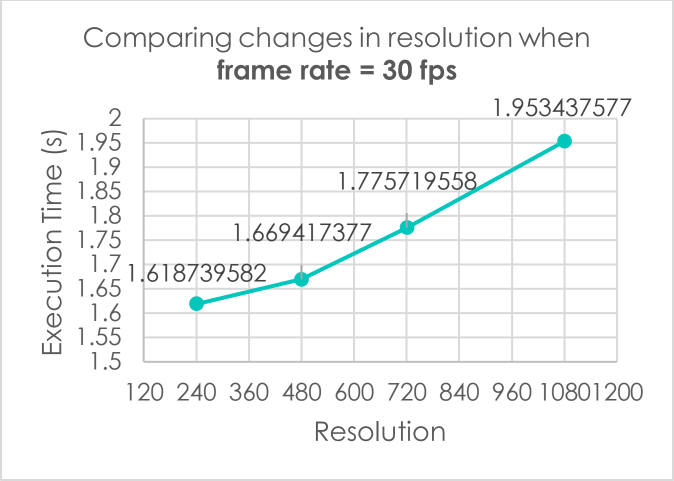

Comparing changes in resolution when **frame rate = 15 fps**

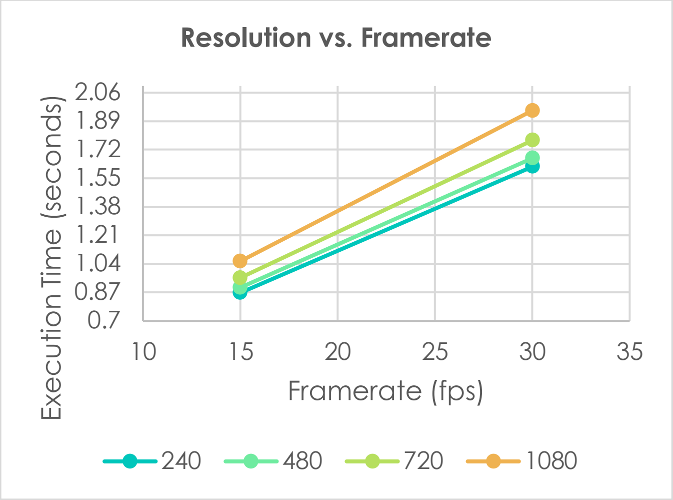

Comparing changes in resolution when **frame rate = 30 fps**



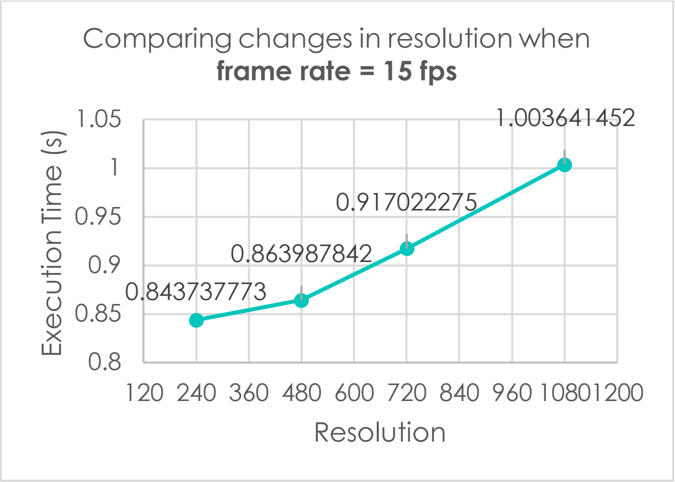
AWS-GPU

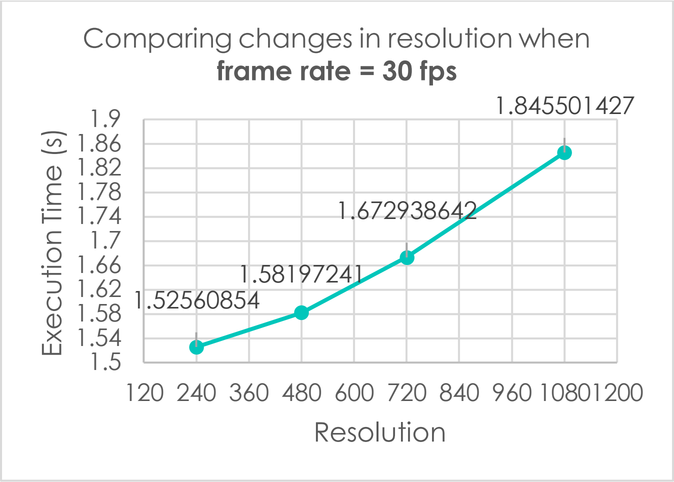


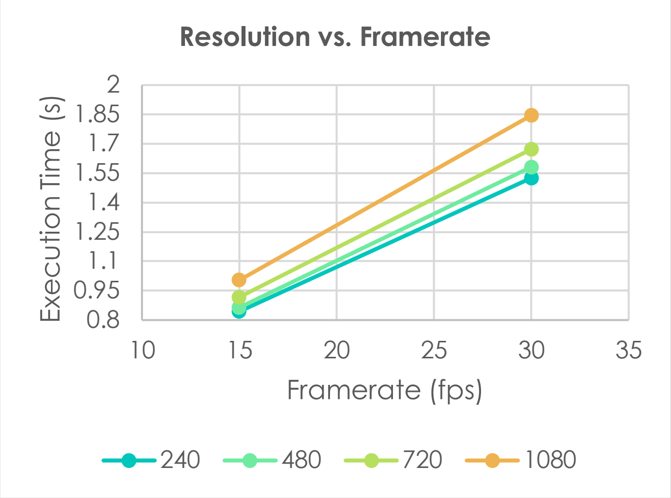




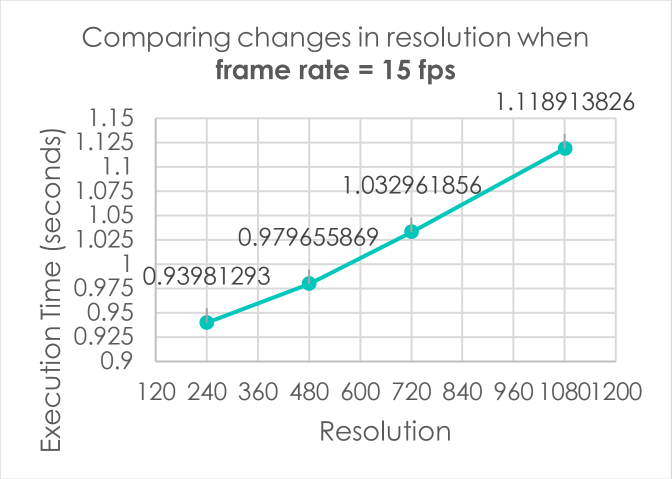
AWS-ML

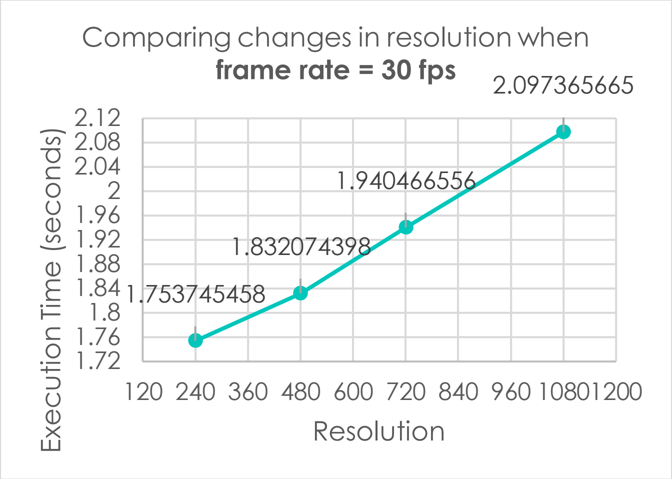


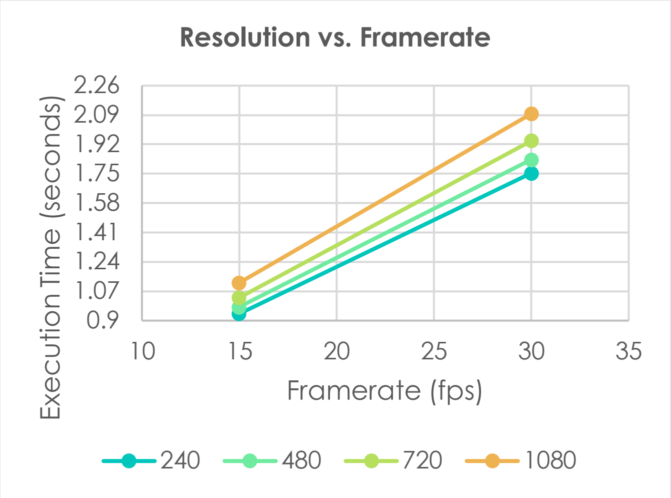




AWS-MO







D. Evaluation

- **CC**
  - Based on the Resolution vs. Framerate graphs, it is evident that there is a linear relationship between increasing resolution and framerate that results in a longer inference time. When the framerate is 15 fps, the fastest inference time is from CC-XLarge with 1.26 seconds. CC-Large and CC-2xLarge follow closely with around 1.33 and 1.34 seconds respectively.
  - Similarly, when the framerate is 30 fps, the lowest inference time is again from CC-XLarge with 2.49 seconds. CC-Large and CC-2xLarge had 2.57 and 2.60 seconds respectively. This indicates that extra CPUs or more memory in the CC-2xLarge, which has 16 CPUs and 32 GB, does not make a difference in performance in comparison to the CC-XLarge that has 8 CPUs and 16 GB. Moreover, this could also show that the CPUs in CC-2xLarge may be allocating resources to some other

process. Overall, if an individual wants to save resources, CC-XLarge would be the best machine type in this situation.

- **AWS**
  - Similarly, there is also a linear relationship in the resolution, framerate, and inference time for the AWS machines. When the framerate is 15 fps, the fastest inference time is from AWS-ML with 0.84 seconds. It is also the fastest when the framerate is 30 fps with 1.52 seconds. Next place in fastest inference time is AWS-GPU, with 0.86 seconds for 15 fps and 1.60 seconds for 30 fps.
  - Since the Fire App utilizes machine learning, it is understandable that the fastest machine type would be AWS-ML that is optimized for machine learning. Moreover, it is reasonable that the next fastest machine type is AWS-GPU because a GPU has its own memory whereas CPUs take up a lot of memory while training a model. However, if an individual wants to save resources, it is better to use the AWS-ML because it costs $0.368 per hour rather than the AWS-GPU where it costs $0.526 per hour in EC2 pricing according to https://aws.amazon.com/ec2/pricing/on-demand/.

III. Video Streaming

A. Project

- Description
  - Video streams, either in the form of on-demand streaming or live streaming, typically have to be converted or transcoded based on the characteristics of clients' devices. Many streaming service providers are becoming reliant on cloud technologies, but there exist challenges in using cloud resources in a cost-efficient manner without any major impact on the quality of the videos. In this project, I benchmarked transcoding a dataset of videos to determine the most time-efficient conversion and machine type.
- Dataset
  - The dataset used in this project includes 100 videos that are split into multiple two-second segments. The videos have been standardized to have a 720p resolution, 30 fps framerate, H.264 codec, and 4500 kb/s bitrate. The dataset can be found here: https://drive.google.com/drive/folders/1KhsxZtC22L-EHoeXmsdmWpkuNZpmS-pL
- Conversion Combinations
  - For this experiment, there are 36 cases to convert the resolution, framerate, and codec of the videos in the dataset. For the H.264 codec, transcoding would only consist of changing the resolution or framerate because the standardized format is H.264 codec:

Resolution	Framerate	Codec	Resolution	Framerate	Codec	Resolution	Framerate	Codec	Resolution	Framerate	Codec
360	24	HEVC	360	24	VP9	360	24	MPEG4	360	24	H.264
360	30	HEVC	360	30	VP9	360	30	MPEG4	360	30	H.264
480	24	HEVC	480	24	VP9	480	24	MPEG4	480	24	H.264
480	30	HEVC	480	30	VP9	480	30	MPEG4	480	30	H.264
720	24	HEVC	720	24	VP9	720	24	MPEG4	720	24	H.264
720	30	HEVC	720	30	VP9	720	30	MPEG4	720	30	H.264
1080	24	HEVC	1080	24	VP9	1080	24	MPEG4	1080	24	H.264
1080	30	HEVC	1080	30	VP9	1080	30	MPEG4	1080	30	H.264
1080	60	HEVC	1080	60	VP9	1080	60	MPEG4	1080	60	H.264

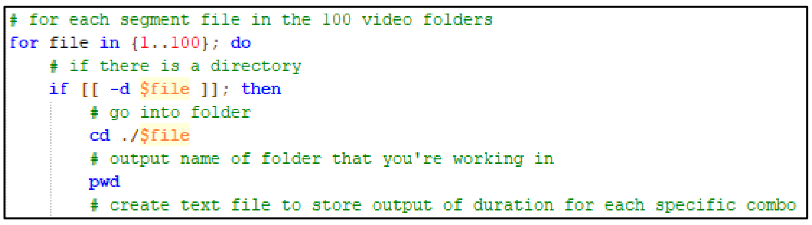
- Machine Types
  - The conversions were benchmarked on several VMs on CC and AWS. I was unable to complete AWS-GPU and the "HEVC script" for the CC-Large instance, which is indicated by a highlighted row. Listed in the tables below are the VMs I was able to get benchmarking results:

Chameleon Cloud				
Type	Abbr.	CPUs	RAM (GB)	Disk (GB)
medium	CC-Medium	2	4	40
large	CC-Large	4	8	40
xlarge	CC-XLarge	8	16	40
xxlarge	CC-2xLarge	16	32	40

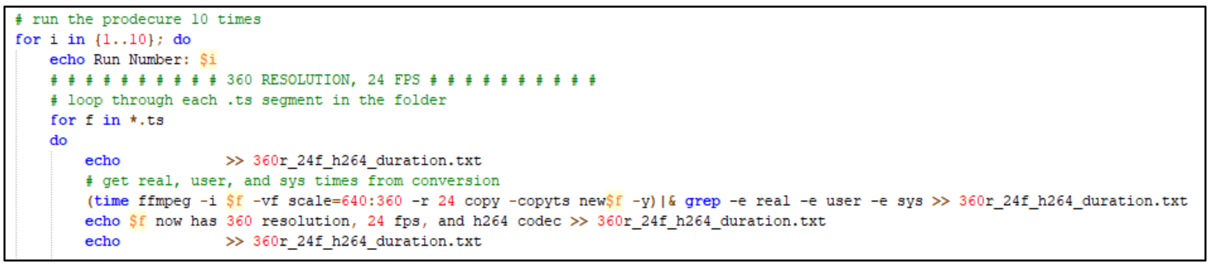
Amazon Web Services					
Type	Family	Abbr.	vCPUs	Memory (GiB)	Instance Storage
c5.2xlarge	Compute optimized	AWS-CO	8	16	EBS only
g4dn.xlarge	GPU instances	AWS-GPU	4	16	1 x 125 (SSD)
m5.2xlarge	General Purpose	AWS-GPI	8 (Intel)	32	EBS only
m5a.2xlarge	General Purpose	AWS-GPA	8 (AMD)	32	EBS only
r5.2xlarge	Memory Optimized	AWS-MO	8	64	EBS only

B. Method

- Scripts
  - For each conversion combination, I wrote four scripts defined by the different codec types. Each script uses FFMPEG and is similar in traversing and transcoding the video folders. FFMPEG is an open-source project consisting of a large collection of libraries and programs that provide the ability to transcode, decode, or encode any media file available. Below are screenshots of some of the "H.264 script":

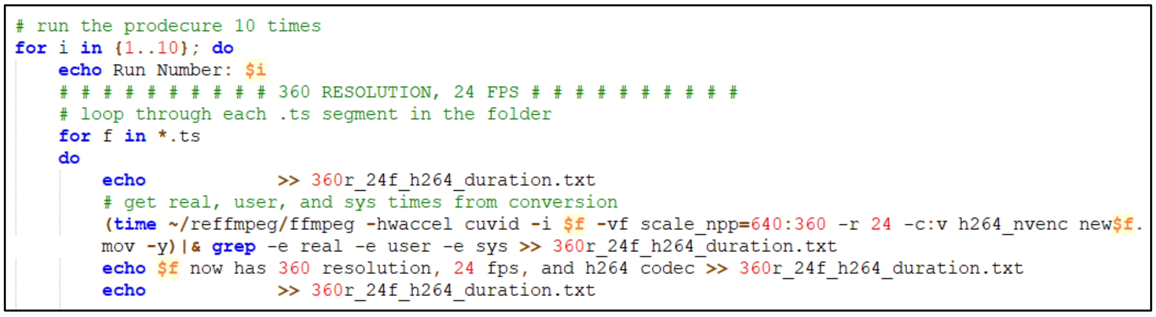


```
# for each segment file in the 100 video folders
for file in {1..100}; do
    # if there is a directory
    if [[ -d $file ]]; then
        # go into folder
        cd ./$file
        # output name of folder that you're working in
        pwd
        # create text file to store output of duration for each specific combo
```



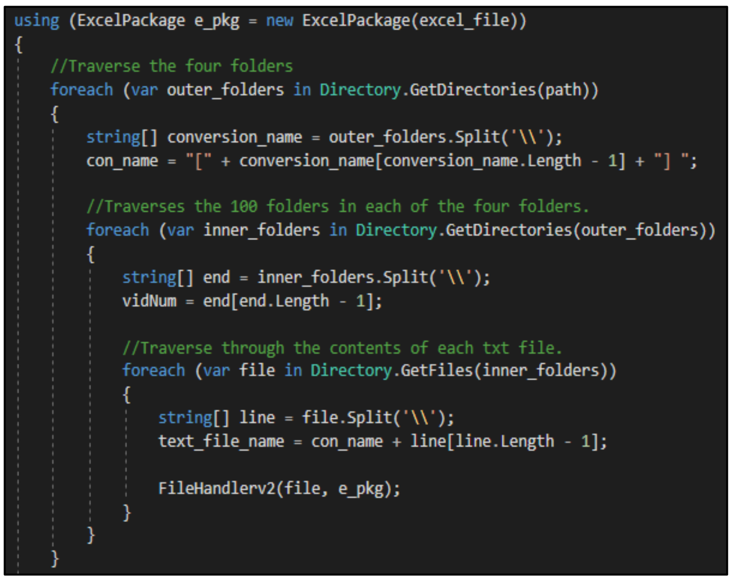
```
# run the prodecure 10 times
for i in {1..10}; do
    echo Run Number: $i
    # # # # # # # # # # 360 RESOLUTION, 24 FPS # # # # # # # # # #
    # loop through each .ts segment in the folder
    for f in *.ts
    do
        echo            >> 360r_24f_h264_duration.txt
        # get real, user, and sys times from conversion
        (time ffmpeg -i $f -vf scale=640:360 -r 24 copy -copyts new$f -y)|& grep -e real -e user -e sys >> 360r_24f_h264_duration.txt
        echo $f now has 360 resolution, 24 fps, and h264 codec >> 360r_24f_h264_duration.txt
        echo            >> 360r_24f_h264_duration.txt
```

  - For *g4dn.xlarge* on AWS that I was unable to complete, the script had to be edited to use the appropriate libraries and utilize the NVidia GPU. Below is how I changed the FFMPEG command according to NVidia's documentation:

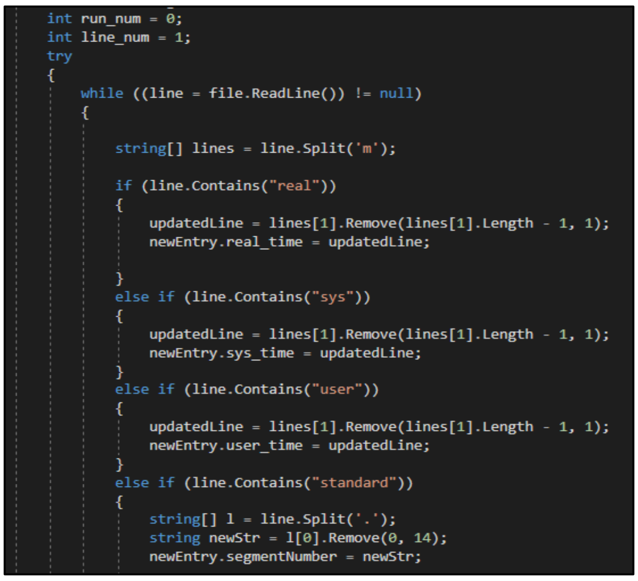


```
# run the prodecure 10 times
for i in {1..10}; do
    echo Run Number: $i
    # # # # # # # # # # 360 RESOLUTION, 24 FPS # # # # # # # # # #
    # loop through each .ts segment in the folder
    for f in *.ts
    do
        echo            >> 360r_24f_h264_duration.txt
        # get real, user, and sys times from conversion
        (time ~/reffmpeg/ffmpeg -hwaccel cuvid -i $f -vf scale_npp=640:360 -r 24 -c:v h264_nvenc new$f.
        mov -y)|& grep -e real -e user -e sys >> 360r_24f_h264_duration.txt
        echo $f now has 360 resolution, 24 fps, and h264 codec >> 360r_24f_h264_duration.txt
        echo            >> 360r_24f_h264_duration.txt
```

- To obtain the execution time of the video conversions, the Linux command "time" provides the real, system, and user times. These times are then extracted from the results into a text file with the Linux command "grep."
- After the benchmarking, I use a C# script to parse and separate the data into an Excel spreadsheet. Below is a sample of my Excel automation script:



```
using (ExcelPackage e_pkg = new ExcelPackage(excel_file))
{
    //Traverse the four folders
    foreach (var outer_folders in Directory.GetDirectories(path))
    {
        string[] conversion_name = outer_folders.Split('\\');
        con_name = "[" + conversion_name[conversion_name.Length - 1] + "] ";

        //Traverses the 100 folders in each of the four folders.
        foreach (var inner_folders in Directory.GetDirectories(outer_folders))
        {
            string[] end = inner_folders.Split('\\');
            vidNum = end[end.Length - 1];

            //Traverse through the contents of each txt file.
            foreach (var file in Directory.GetFiles(inner_folders))
            {
                string[] line = file.Split('\\');
                text_file_name = con_name + line[line.Length - 1];

                FileHandlerv2(file, e_pkg);
            }
        }
    }
```



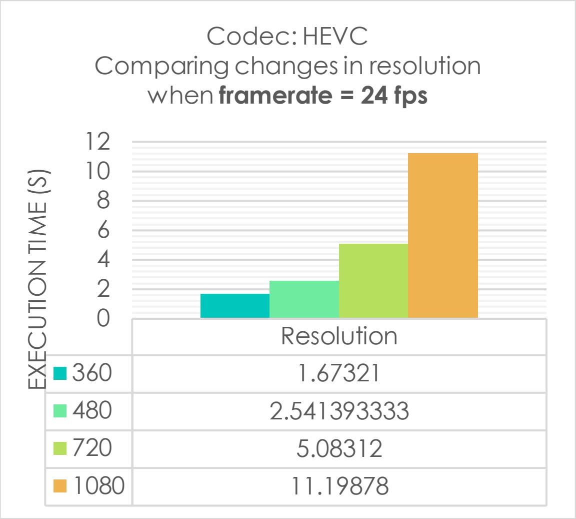
```
    int run_num = 0;
    int line_num = 1;
    try
    {
        while ((line = file.ReadLine()) != null)
        {

            string[] lines = line.Split('m');

            if (line.Contains("real"))
            {
                updatedLine = lines[1].Remove(lines[1].Length - 1, 1);
                newEntry.real_time = updatedLine;

            }
            else if (line.Contains("sys"))
            {
                updatedLine = lines[1].Remove(lines[1].Length - 1, 1);
                newEntry.sys_time = updatedLine;
            }
            else if (line.Contains("user"))
            {
                updatedLine = lines[1].Remove(lines[1].Length - 1, 1);
                newEntry.user_time = updatedLine;
            }
            else if (line.Contains("standard"))
            {
                string[] l = line.Split('.');
                string newStr = l[0].Remove(0, 14);
                newEntry.segmentNumber = newStr;
```

- Experiment Procedure
  - I used my 4 scripts on the 8 VMs to obtain real, system, and user time for the 36 conversion cases a total of 10 runs.
  - The data is exported to text files where I then parse to an Excel file to analyze the output using my Excel Automation script mentioned above.
- Analysis Procedure
  - I used a random number generator to choose 30 different video numbers and again for the segments. I only chose videos that have a 2-second duration. With these videos, I compare the changes in framerate, resolution, and codec with different constants. Below is the list of videos I have analyzed for each VM:

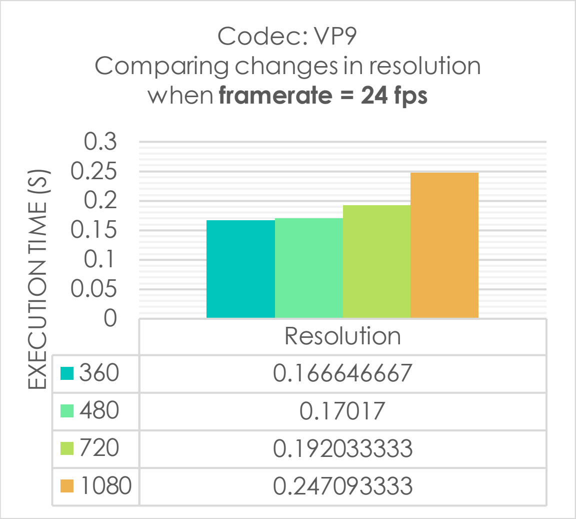
Video Number	Segment
1	26
2	8
3	1
5	3
8	5
10	2
15	9
16	3
19	2
20	11
22	16
26	6
35	4
42	3
43	1
47	5
50	4
51	3
54	6
56	12
58	4
62	72
63	16
64	61
65	2
68	9
74	7
77	12
78	10
82	1

C. Experiment Results

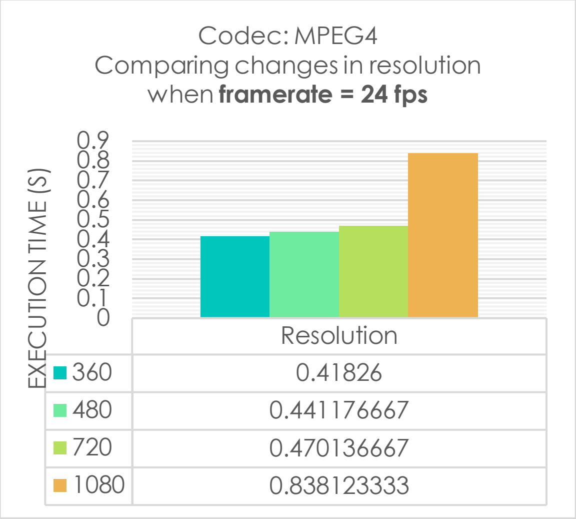
CC-Medium: Framerate as the constant



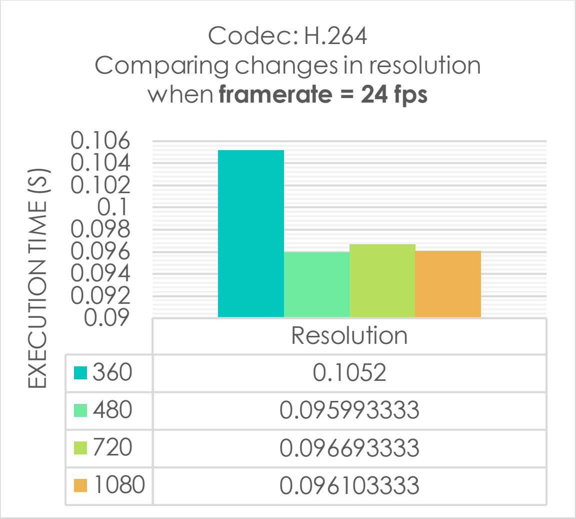
	Resolution
■ 360	1.67321
■ 480	2.541393333
■ 720	5.08312
■ 1080	11.19878



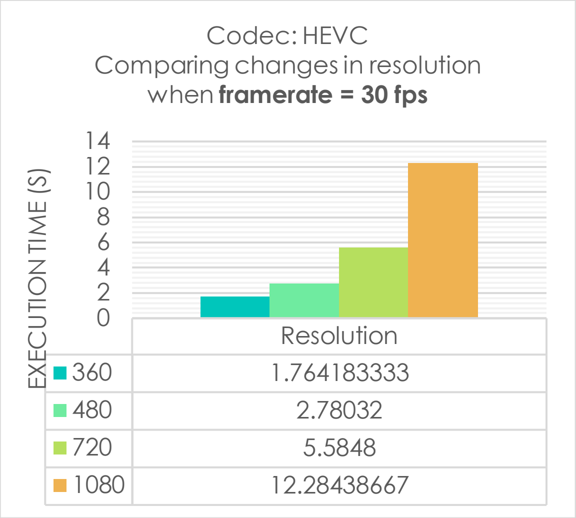
	Resolution
■ 360	0.166646667
■ 480	0.17017
■ 720	0.192033333
■ 1080	0.247093333



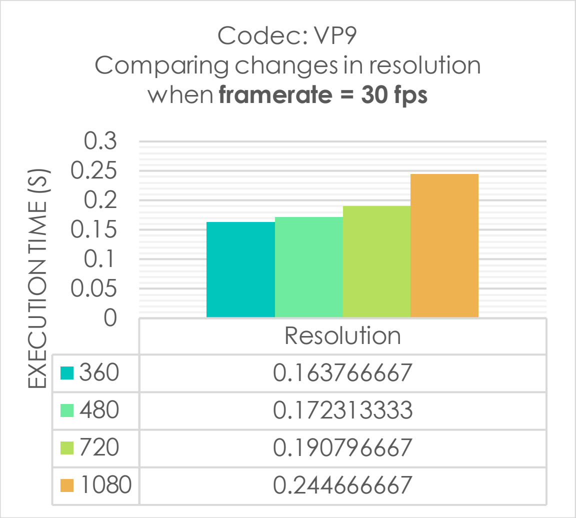
	Resolution
■ 360	0.41826
■ 480	0.441176667
■ 720	0.470136667
■ 1080	0.838123333



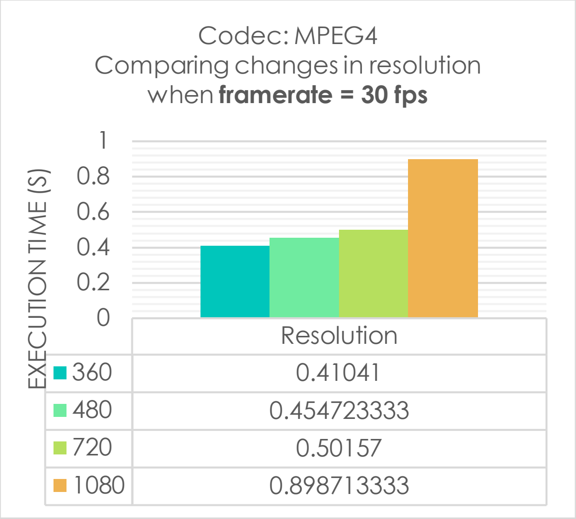
	Resolution
■ 360	0.1052
■ 480	0.095993333
■ 720	0.096693333
■ 1080	0.096103333



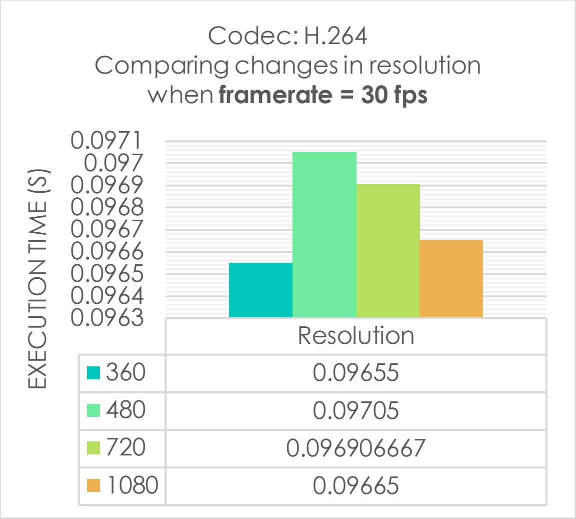
	Resolution
■ 360	1.764183333
■ 480	2.78032
■ 720	5.5848
■ 1080	12.28438667



	Resolution
■ 360	0.163766667
■ 480	0.172313333
■ 720	0.190796667
■ 1080	0.244666667

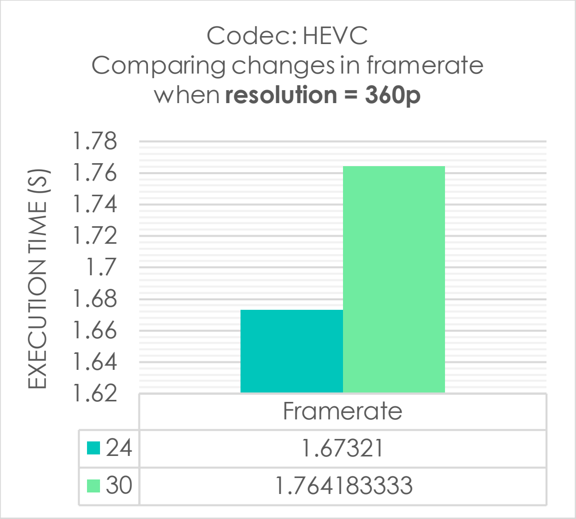


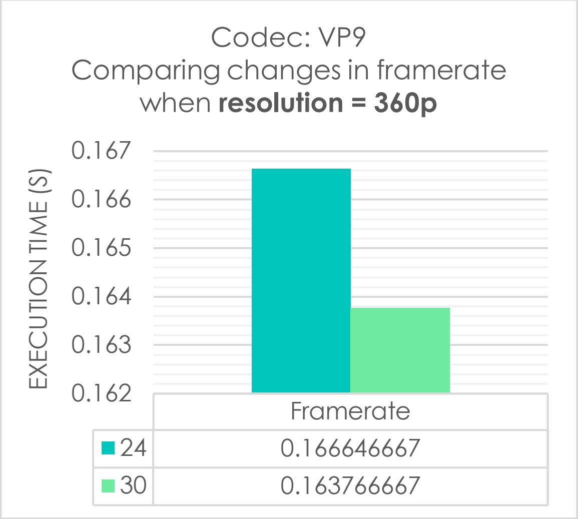
	Resolution
■ 360	0.41041
■ 480	0.454723333
■ 720	0.50157
■ 1080	0.898713333

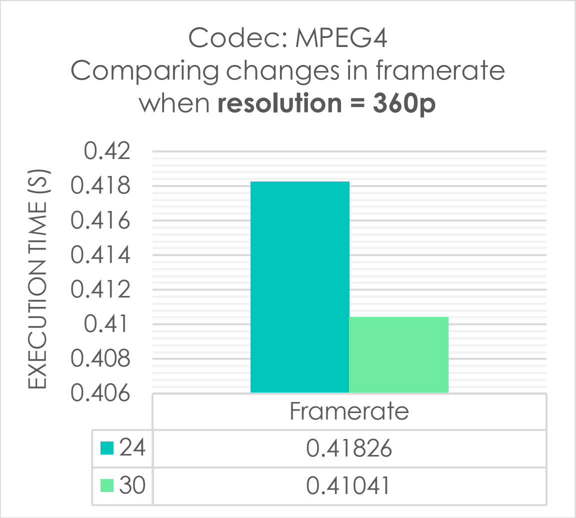


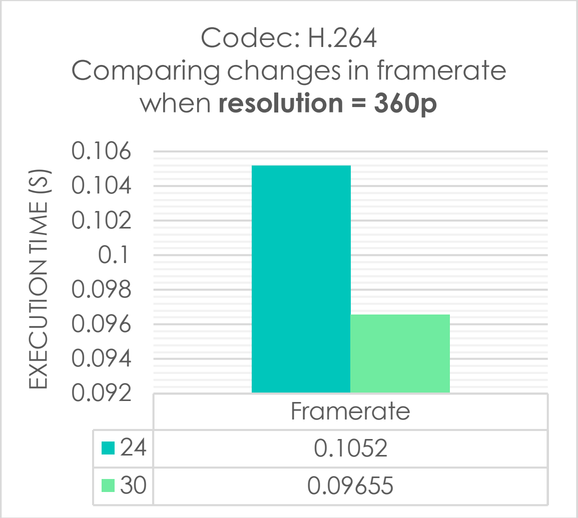
	Resolution
■ 360	0.09655
■ 480	0.09705
■ 720	0.096906667
■ 1080	0.09665

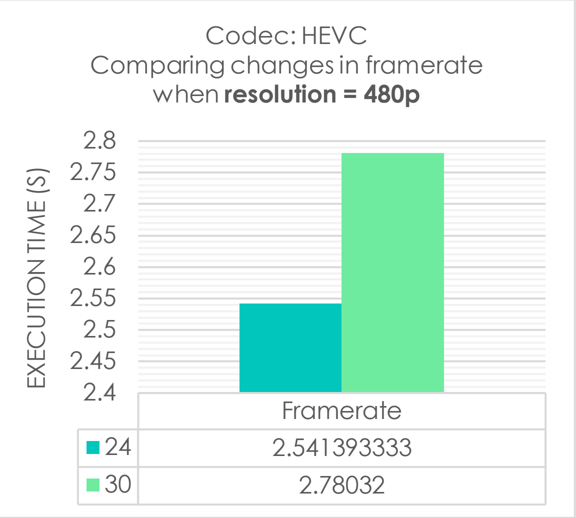
CC-Medium: Resolution as the constant

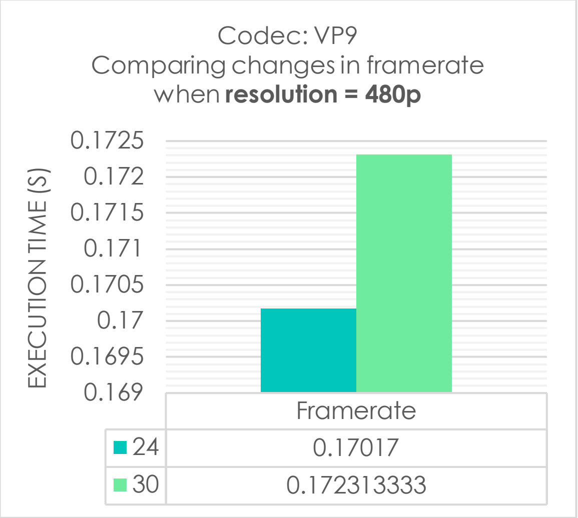


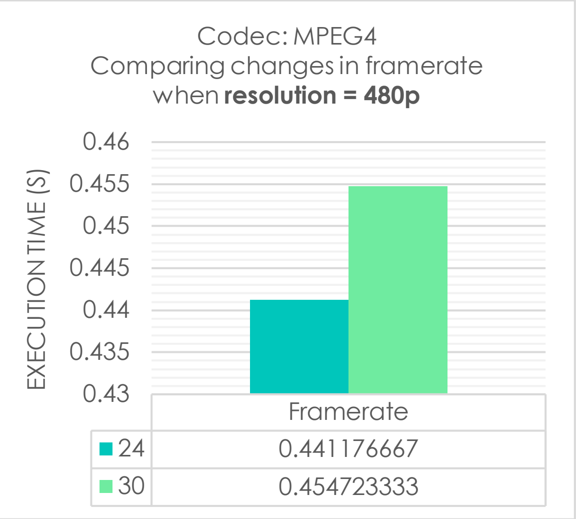


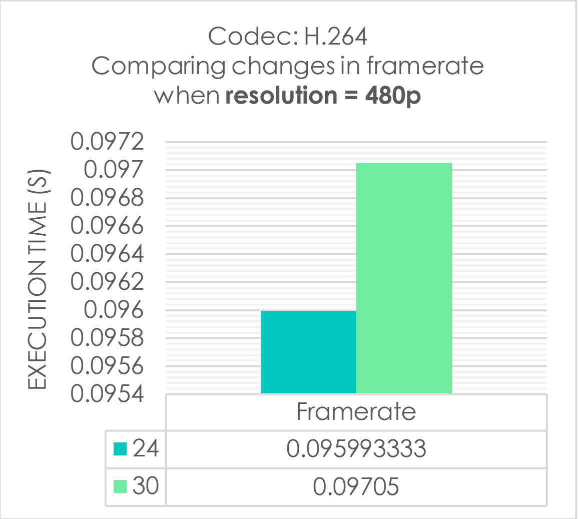


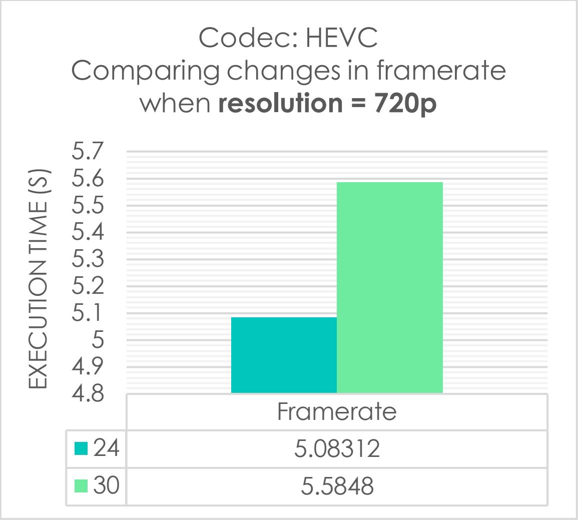


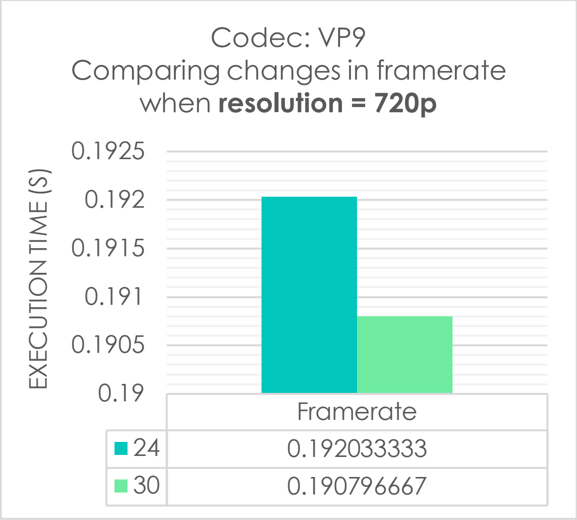


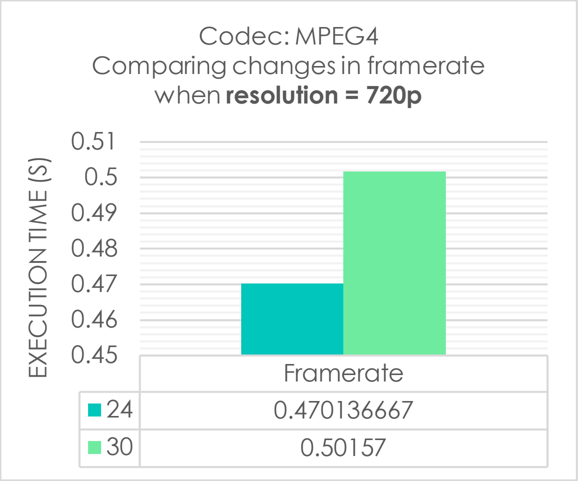


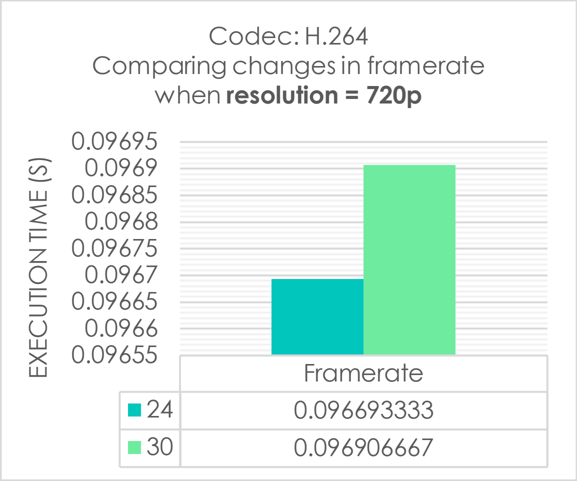


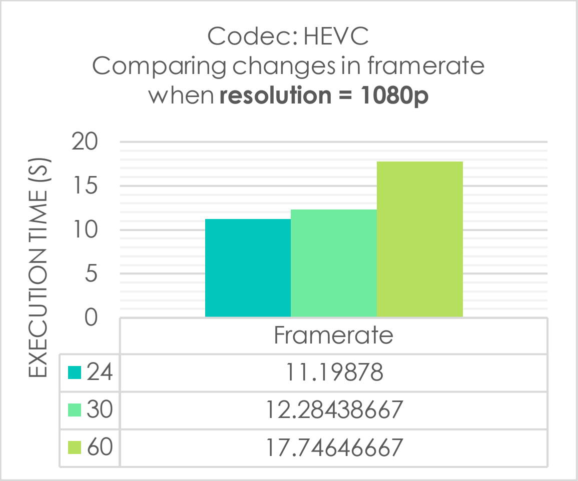


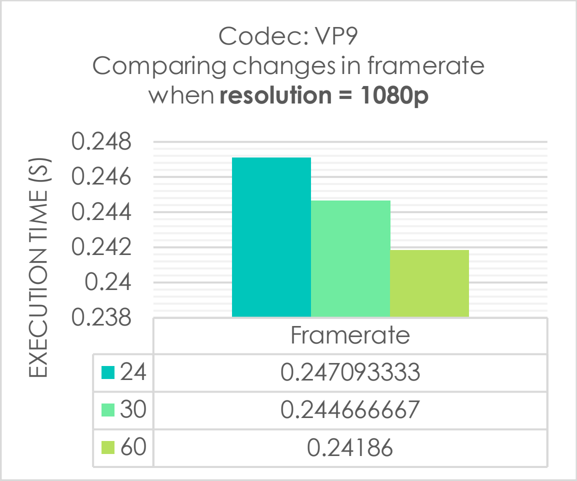

Codec: HEVC
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
5.7
5.6
5.5
5.4
5.3
5.2
5.1
5
4.9
4.8
Framerate
24
5.08312
30
5.5848

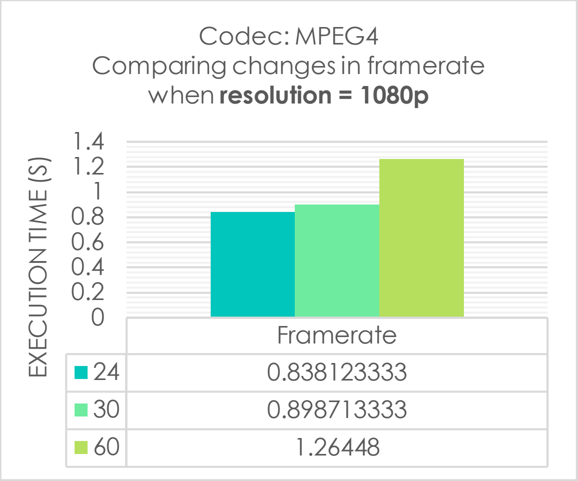

Codec: VP9
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.1925
0.192
0.1915
0.191
0.1905
0.19
Framerate
24
0.192033333
30
0.190796667

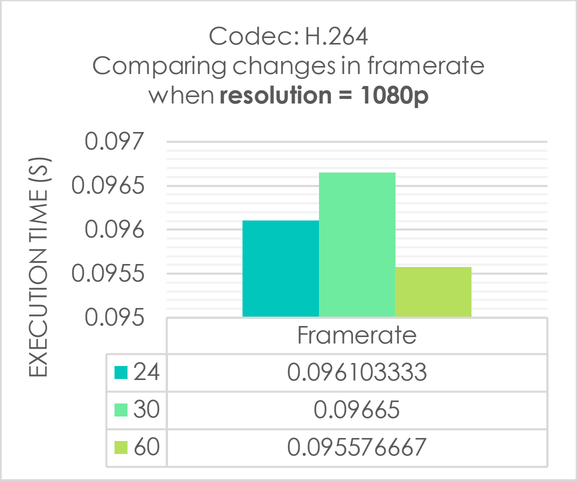

Codec: MPEG4
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.51
0.5
0.49
0.48
0.47
0.46
0.45
Framerate
24
0.470136667
30
0.50157

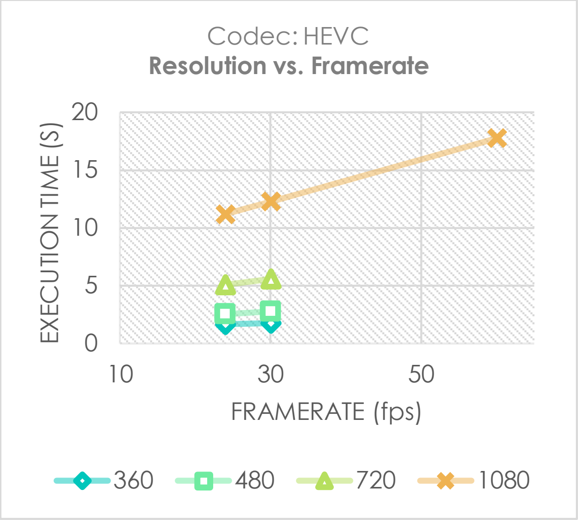

Codec: H.264
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.09695
0.0969
0.09685
0.0968
0.09675
0.0967
0.09665
0.0966
0.09655
Framerate
24
0.096693333
30
0.096906667

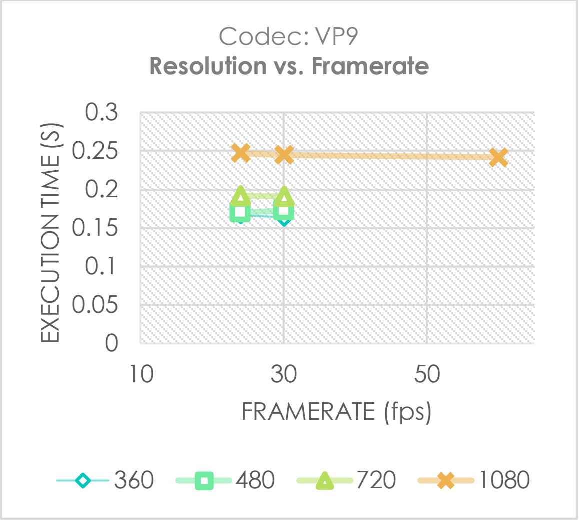

Codec: HEVC
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
20
15
10
5
0
Framerate
24
11.19878
30
12.28438667
60
17.74646667

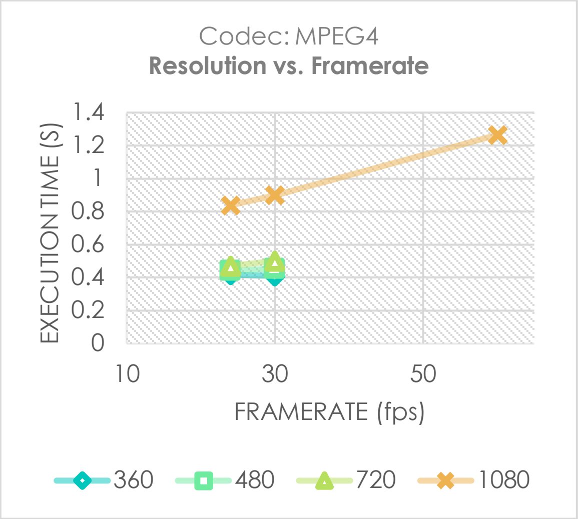

Codec: VP9
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.248
0.246
0.244
0.242
0.24
0.238
Framerate
24
0.247093333
30
0.244666667
60
0.24186

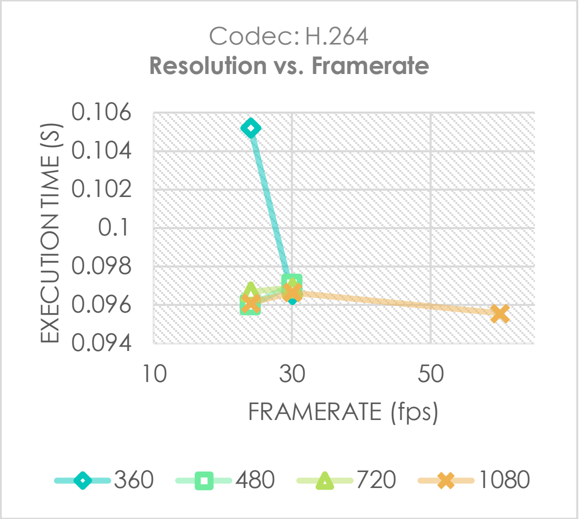

Codec: MPEG4
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Framerate
24
0.838123333
30
0.898713333
60
1.26448

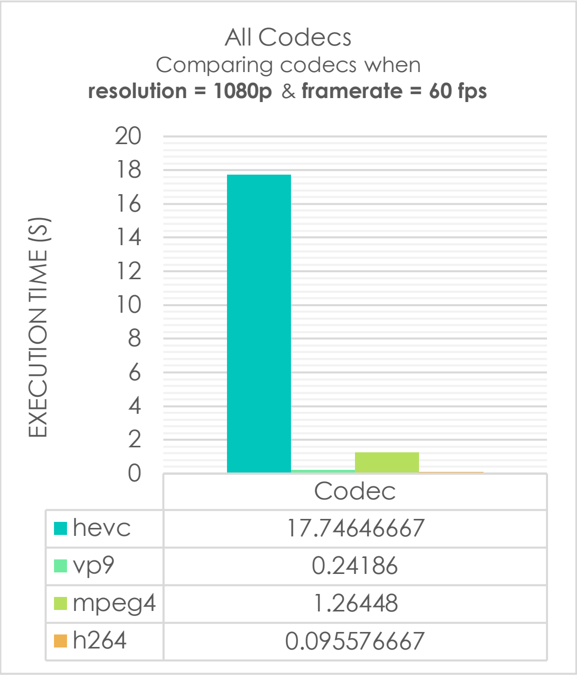

Codec: H.264
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.097
0.0965
0.096
0.0955
0.095
Framerate
24
0.096103333
30
0.09665
60
0.095576667



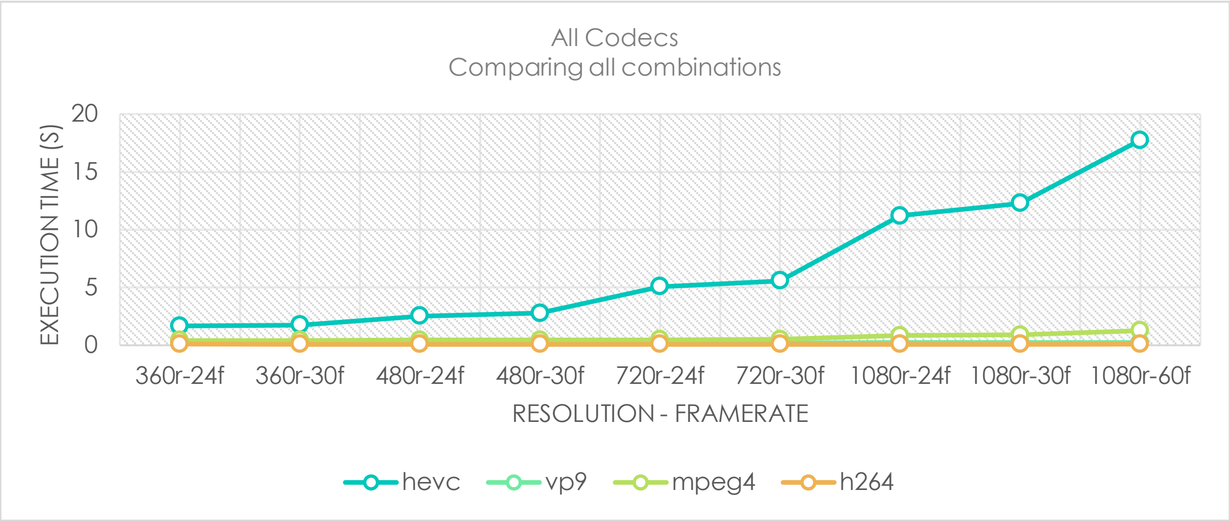




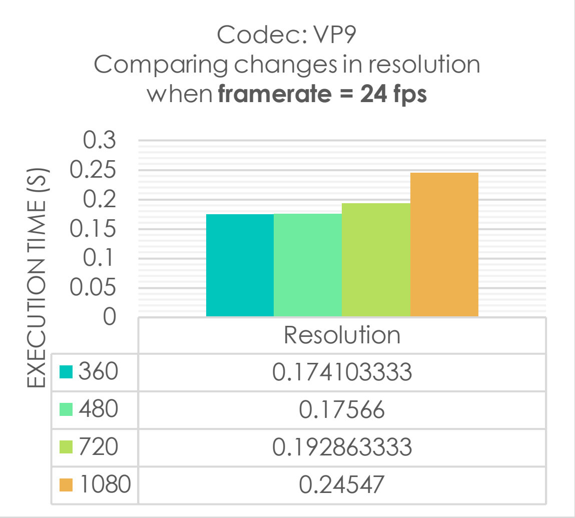




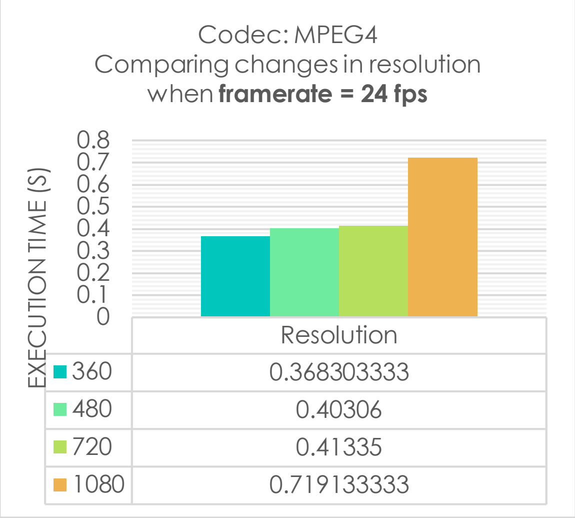
	Codec
hevc	17.74646667
vp9	0.24186
mpeg4	1.26448
h264	0.095576667



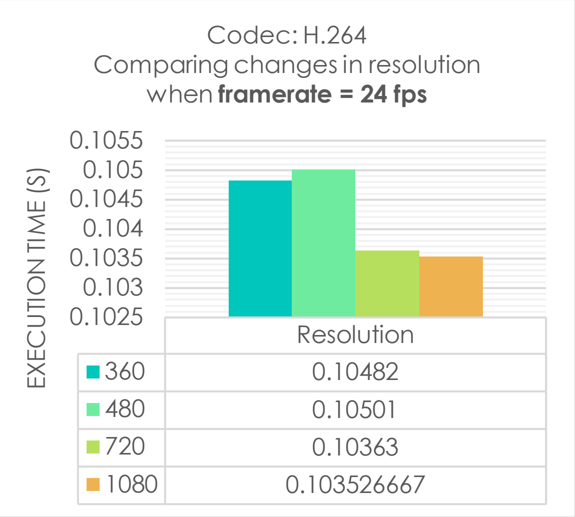
CC-Large: Framerate as the constant


Codec: VP9
Comparing changes in resolution
when **framerate = 24 fps**

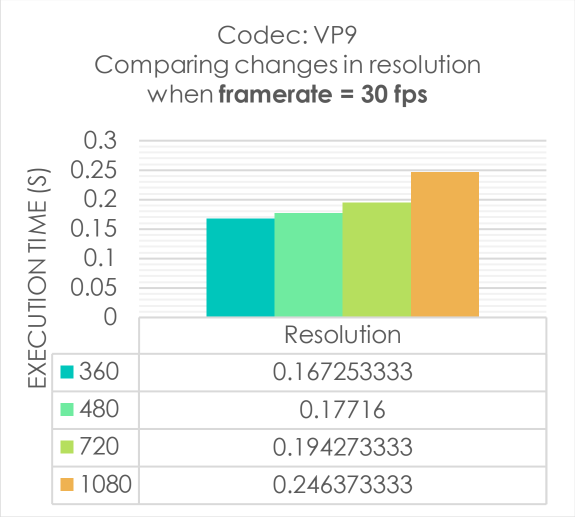
	Resolution
■ 360	0.174103333
■ 480	0.17566
■ 720	0.192863333
■ 1080	0.24547


Codec: MPEG4
Comparing changes in resolution
when **framerate = 24 fps**

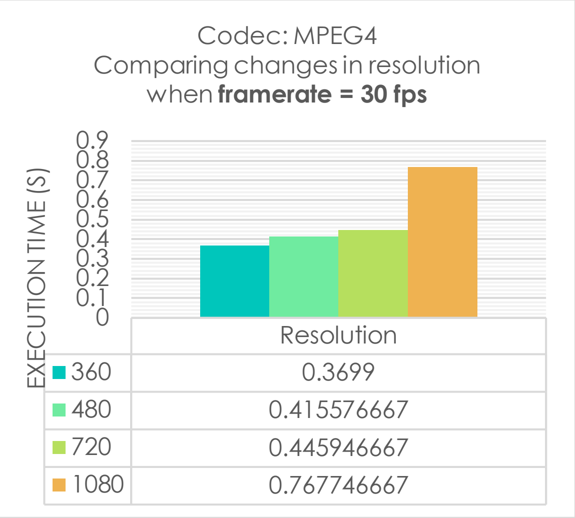
	Resolution
■ 360	0.368303333
■ 480	0.40306
■ 720	0.41335
■ 1080	0.719133333


Codec: H.264
Comparing changes in resolution
when **framerate = 24 fps**

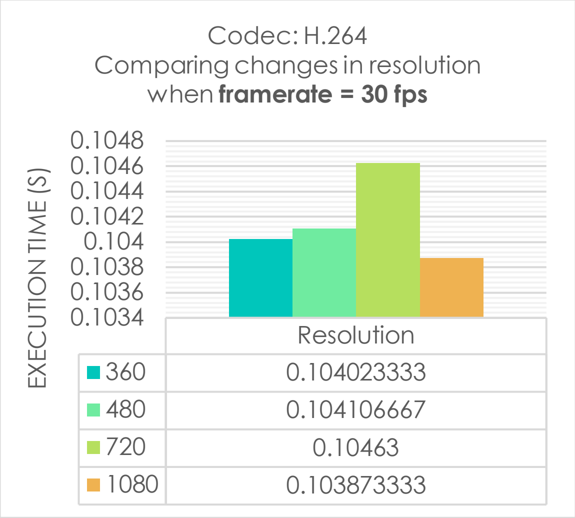
	Resolution
■ 360	0.10482
■ 480	0.10501
■ 720	0.10363
■ 1080	0.103526667


Codec: VP9
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.167253333
■ 480	0.17716
■ 720	0.194273333
■ 1080	0.246373333

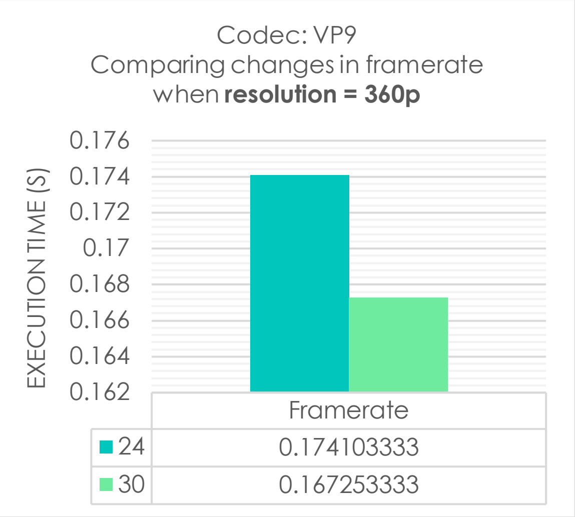

Codec: MPEG4
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.3699
■ 480	0.415576667
■ 720	0.445946667
■ 1080	0.767746667

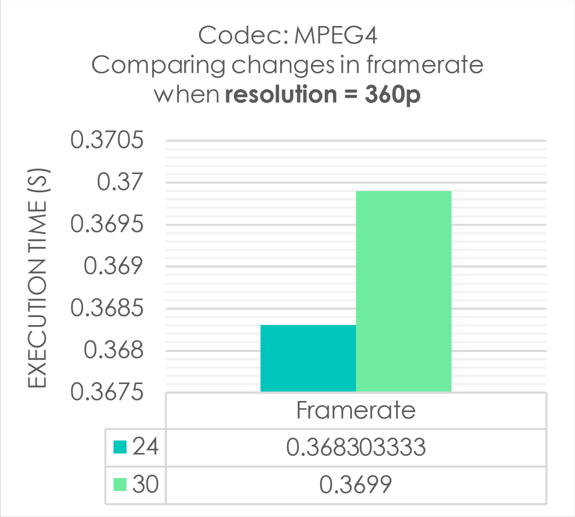

Codec: H.264
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.104023333
■ 480	0.104106667
■ 720	0.10463
■ 1080	0.103873333

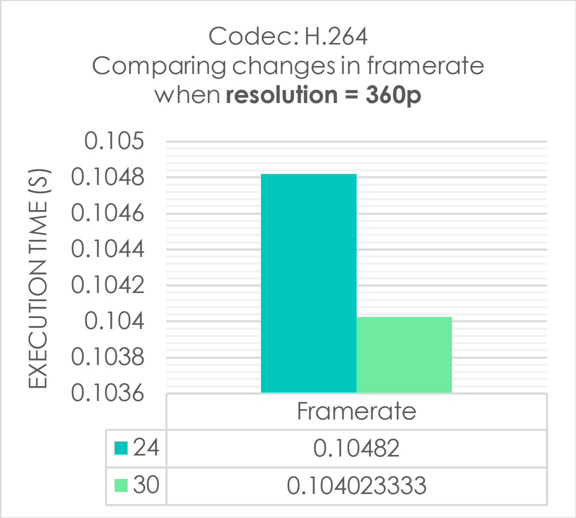
CC-Large: Resolution as the constant

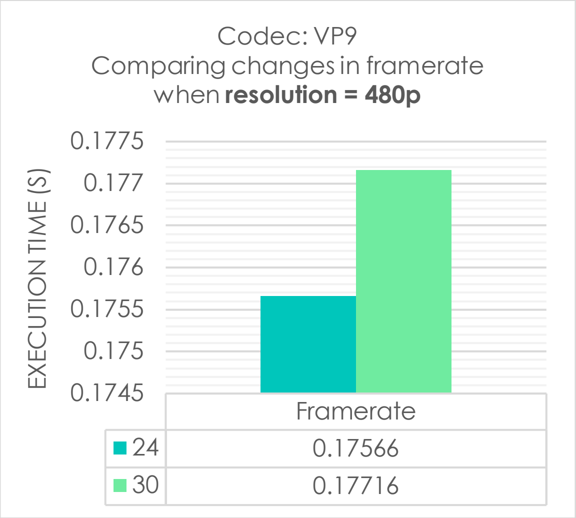

Codec: VP9
Comparing changes in framerate
when **resolution = 360p**

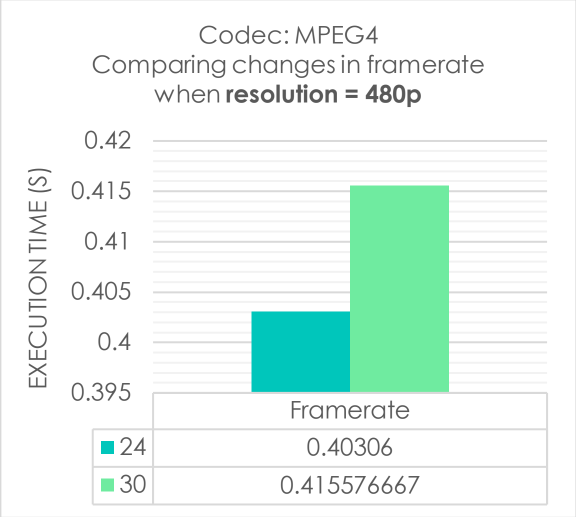
	Framerate
■ 24	0.174103333
■ 30	0.167253333

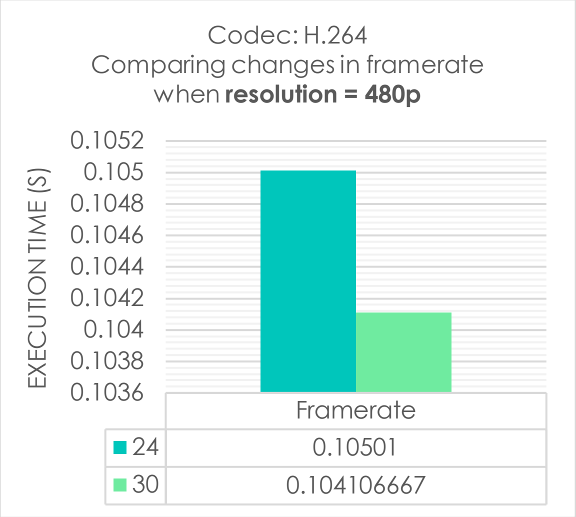

Codec: MPEG4
Comparing changes in framerate
when **resolution = 360p**

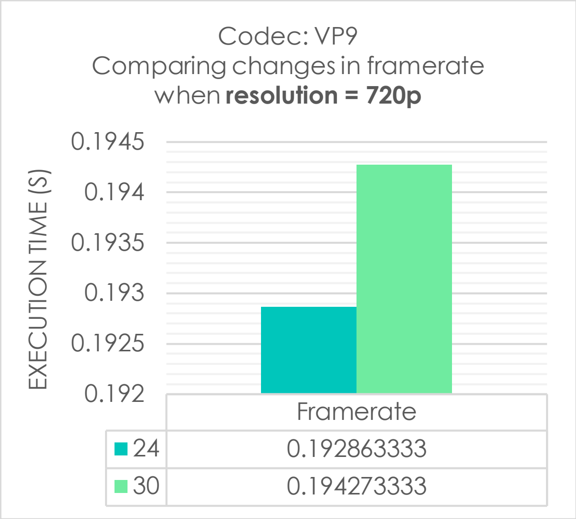
	Framerate
■ 24	0.368303333
■ 30	0.3699

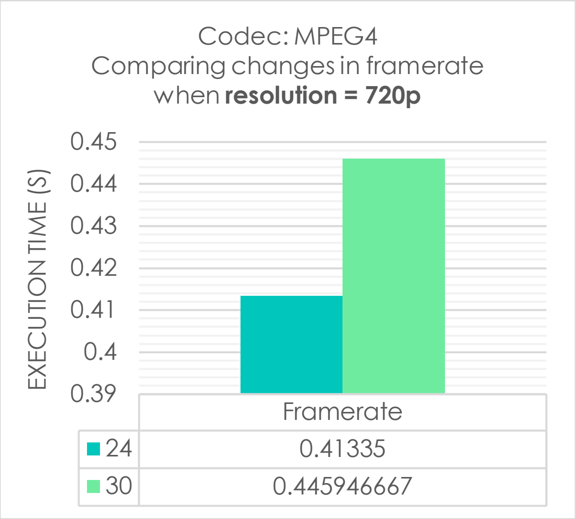

Codec: H.264
Comparing changes in framerate
when **resolution = 360p**

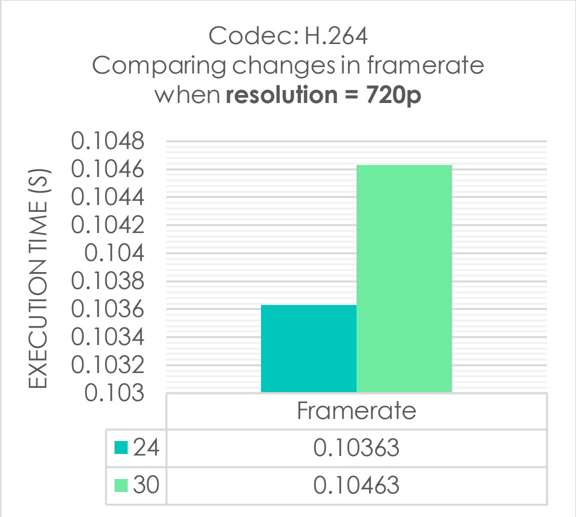

Codec: VP9
Comparing changes in framerate
when **resolution = 480p**

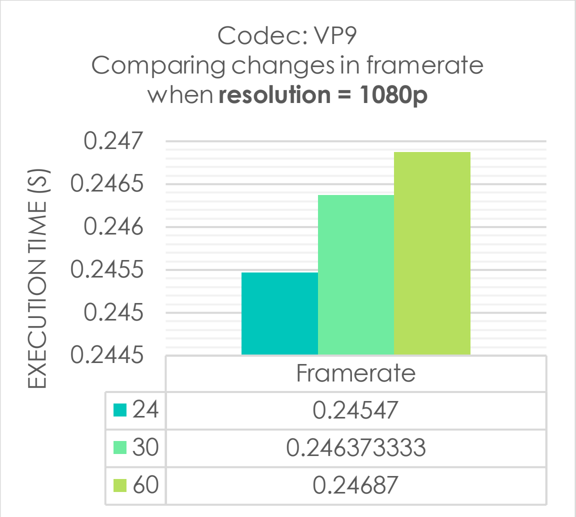

Codec: MPEG4
Comparing changes in framerate
when **resolution = 480p**

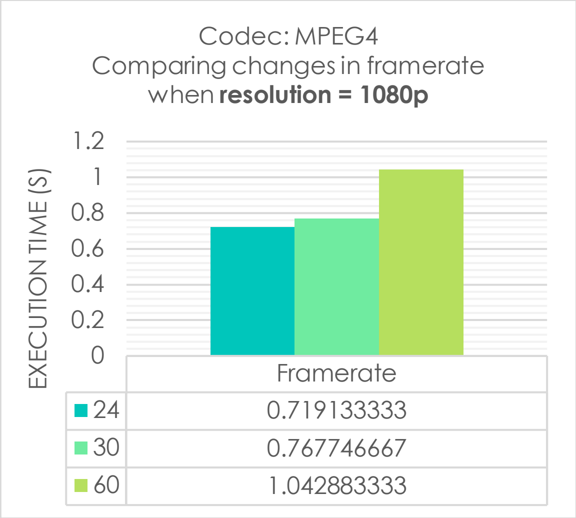

Codec: H.264
Comparing changes in framerate
when **resolution = 480p**


Codec: VP9
Comparing changes in framerate
when **resolution = 720p**

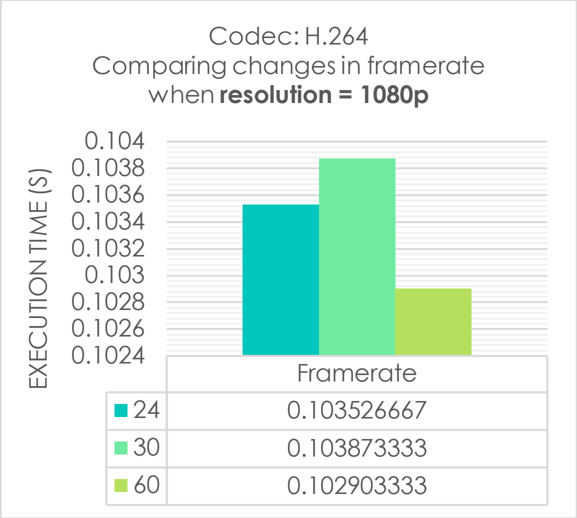

Codec: MPEG4
Comparing changes in framerate
when **resolution = 720p**


Codec: H.264
Comparing changes in framerate
when **resolution = 720p**

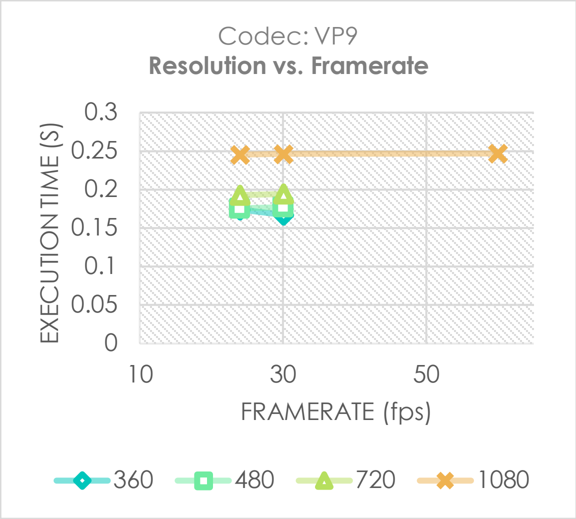

Codec: VP9
Comparing changes in framerate
when **resolution = 1080p**

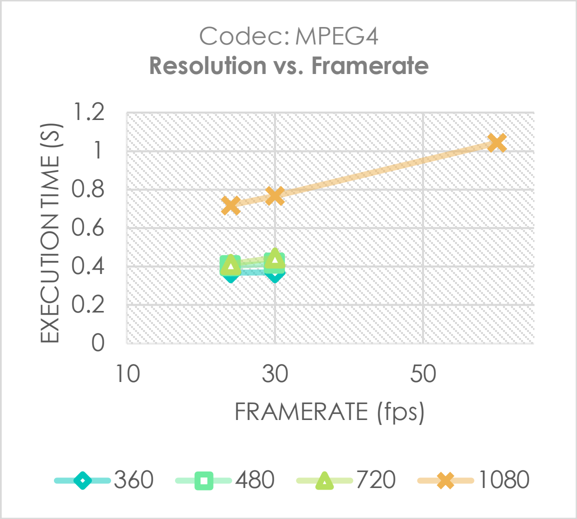

Codec: MPEG4
Comparing changes in framerate
when **resolution = 1080p**

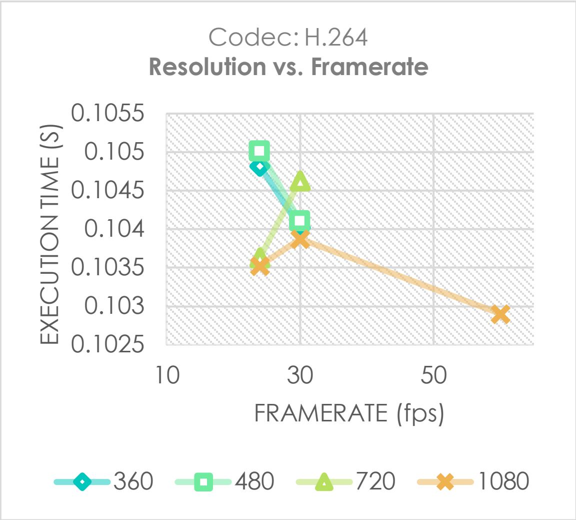
	Framerate
■ 24	0.719133333
■ 30	0.767746667
■ 60	1.042883333

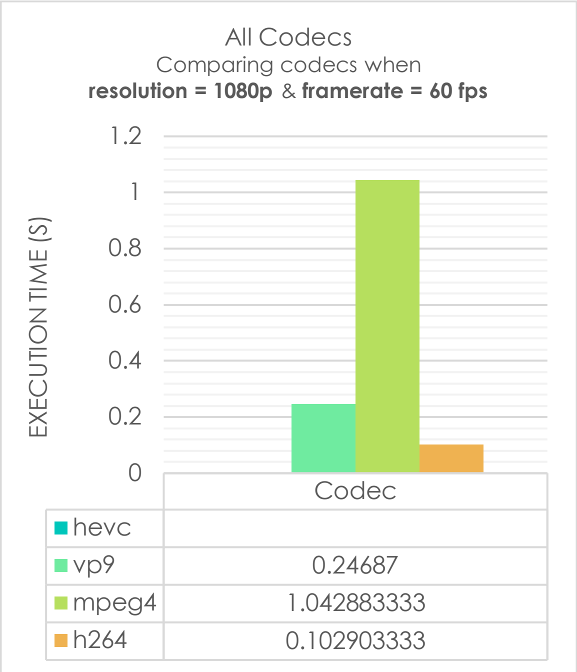

Codec: H.264
Comparing changes in framerate
when **resolution = 1080p**

	Framerate
■ 24	0.103526667
■ 30	0.103873333
■ 60	0.102903333

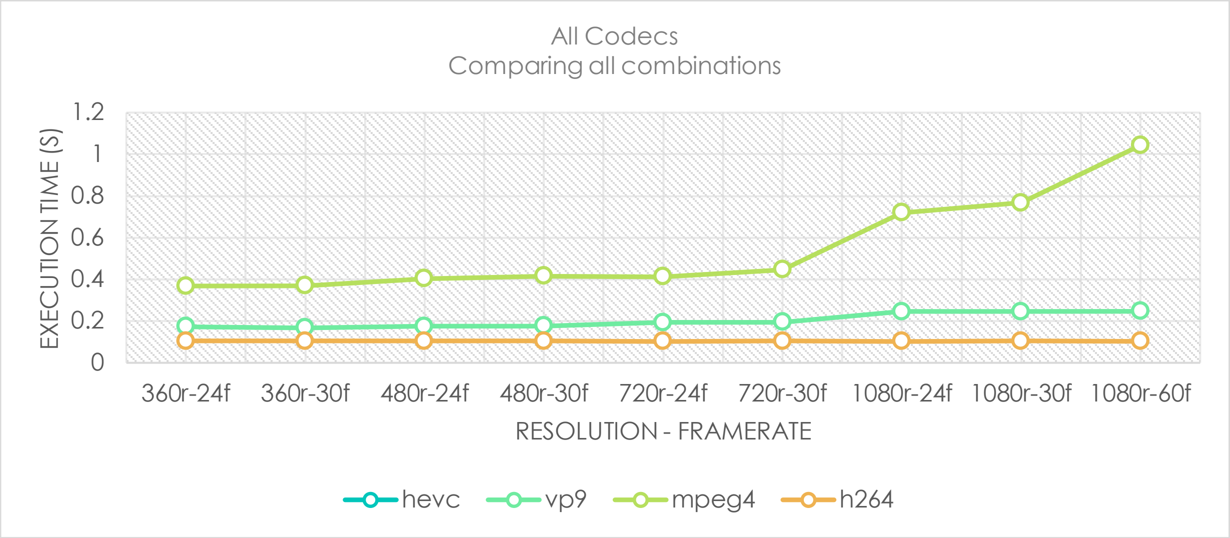

Codec: VP9
Resolution vs. Framerate


Codec: MPEG4
Resolution vs. Framerate

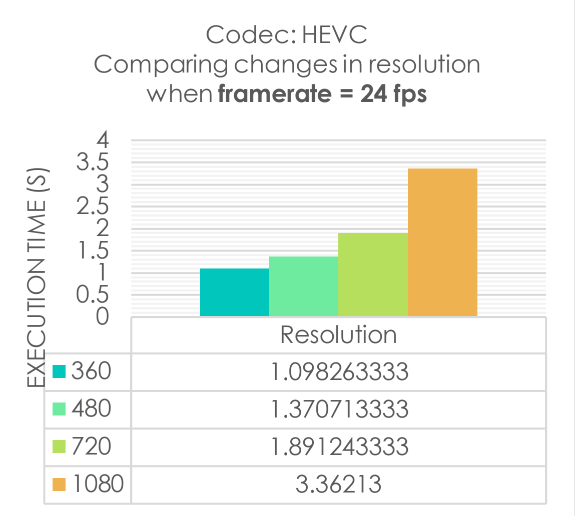

Codec: H.264
Resolution vs. Framerate


All Codecs
Comparing codecs when
resolution = 1080p & **framerate = 60 fps**

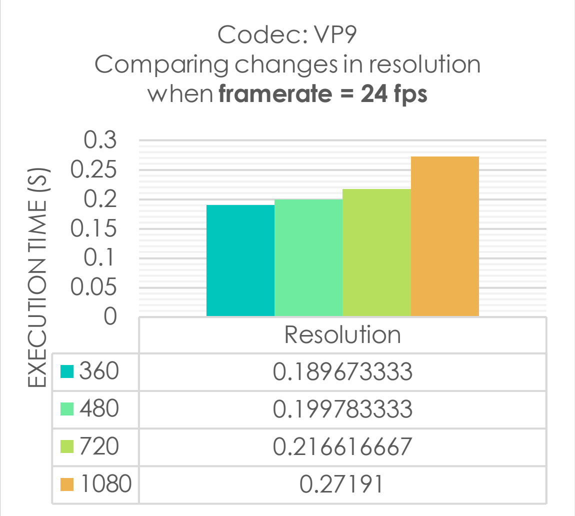
	Codec
■ hevc	
■ vp9	0.24687
■ mpeg4	1.042883333
■ h264	0.102903333


All Codecs
Comparing all combinations

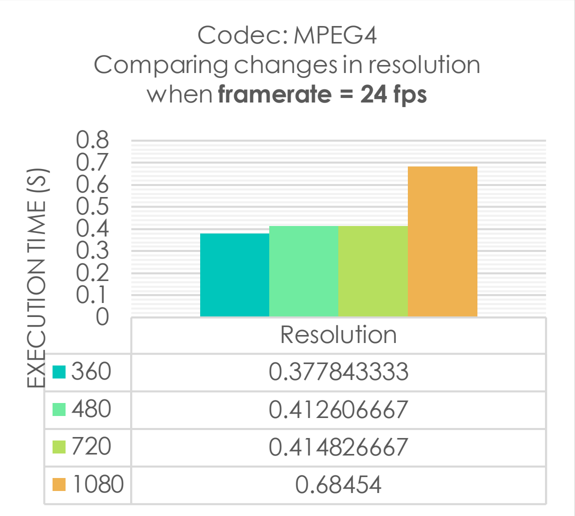
CC-XLarge: Framerate as the constant


Codec: HEVC
Comparing changes in resolution
when **framerate = 24 fps**

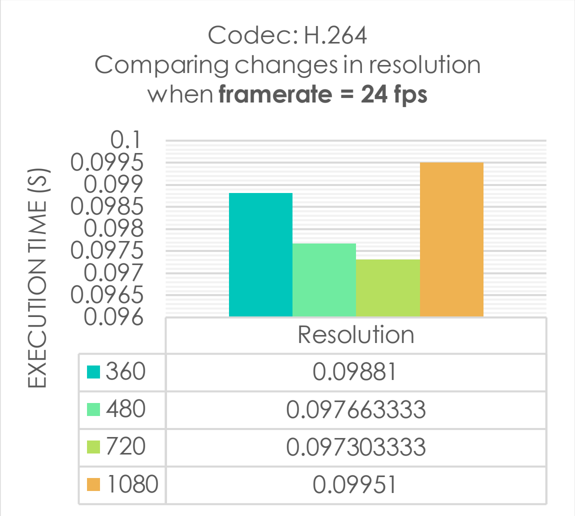
	Resolution
■ 360	1.098263333
■ 480	1.370713333
■ 720	1.891243333
■ 1080	3.36213


Codec: VP9
Comparing changes in resolution
when **framerate = 24 fps**

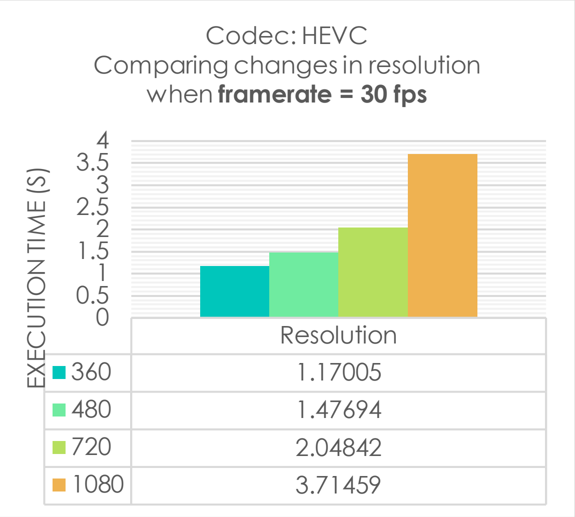
	Resolution
■ 360	0.189673333
■ 480	0.199783333
■ 720	0.216616667
■ 1080	0.27191


Codec: MPEG4
Comparing changes in resolution
when **framerate = 24 fps**

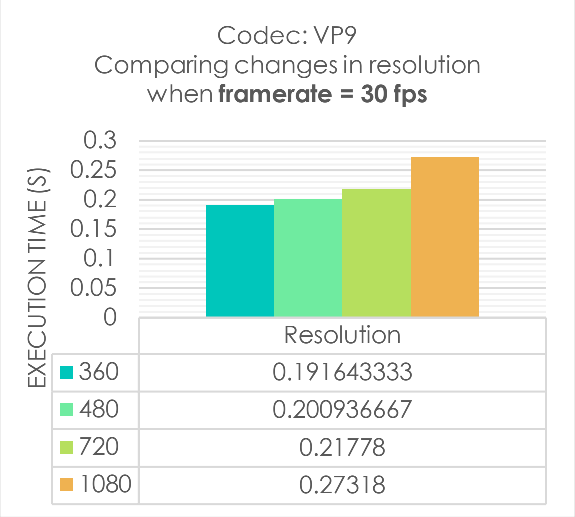
	Resolution
■ 360	0.377843333
■ 480	0.412606667
■ 720	0.414826667
■ 1080	0.68454


Codec: H.264
Comparing changes in resolution
when **framerate = 24 fps**

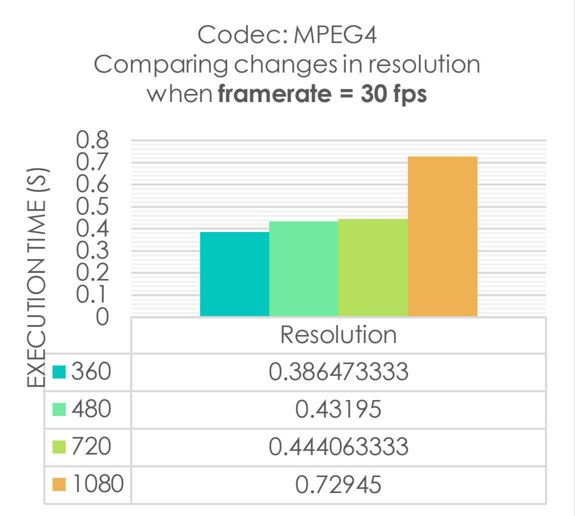
	Resolution
■ 360	0.09881
■ 480	0.097663333
■ 720	0.097303333
■ 1080	0.09951


Codec: HEVC
Comparing changes in resolution
when **framerate = 30 fps**

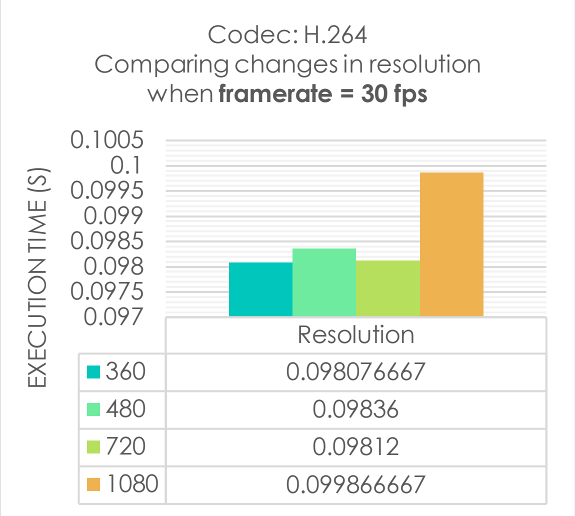
	Resolution
■ 360	1.17005
■ 480	1.47694
■ 720	2.04842
■ 1080	3.71459


Codec: VP9
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.191643333
■ 480	0.200936667
■ 720	0.21778
■ 1080	0.27318

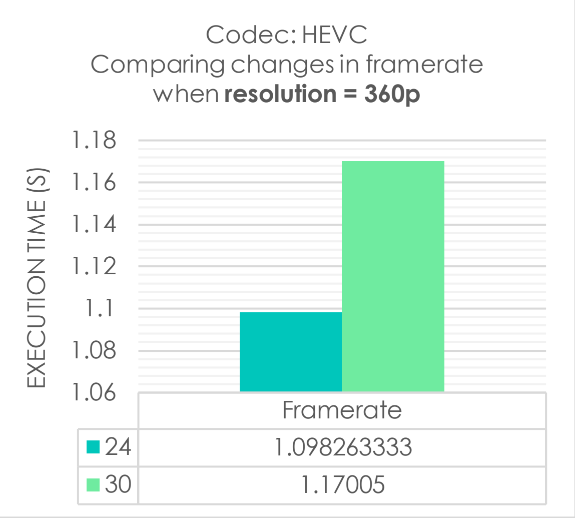

Codec: MPEG4
Comparing changes in resolution
when **framerate = 30 fps**

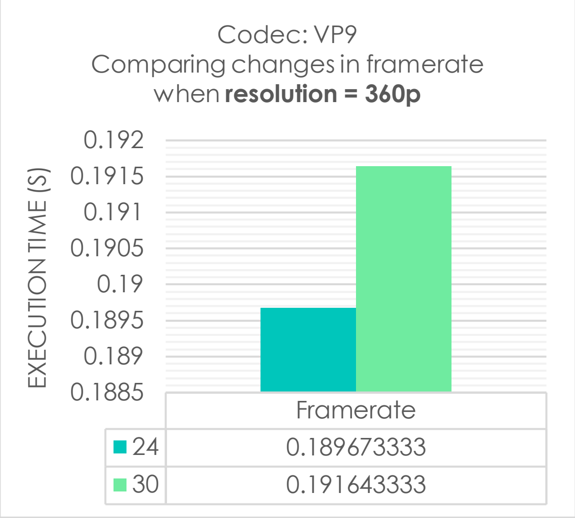
	Resolution
■ 360	0.386473333
■ 480	0.43195
■ 720	0.444063333
■ 1080	0.72945

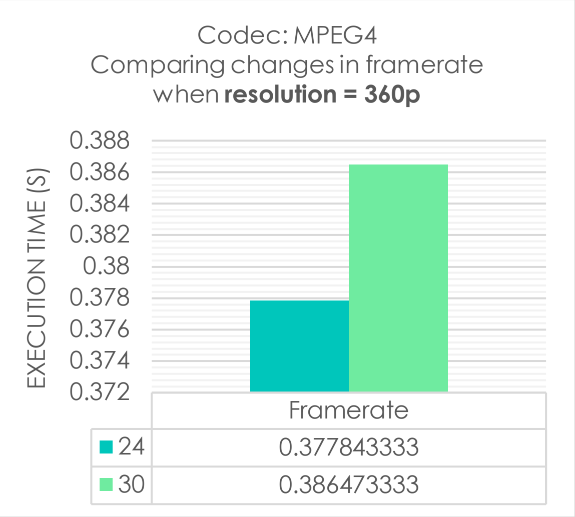

Codec: H.264
Comparing changes in resolution
when **framerate = 30 fps**

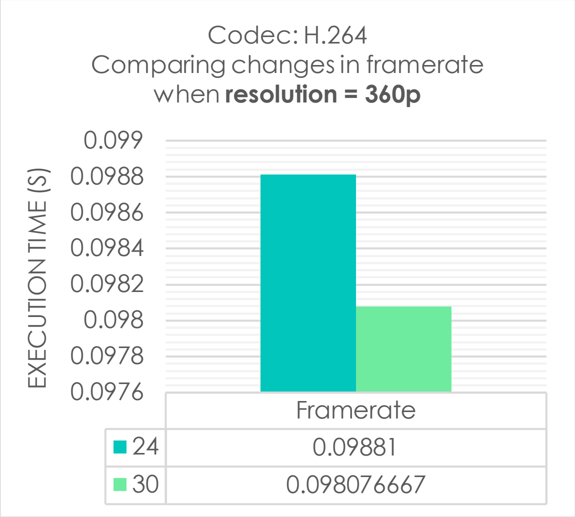
	Resolution
■ 360	0.098076667
■ 480	0.09836
■ 720	0.09812
■ 1080	0.099866667

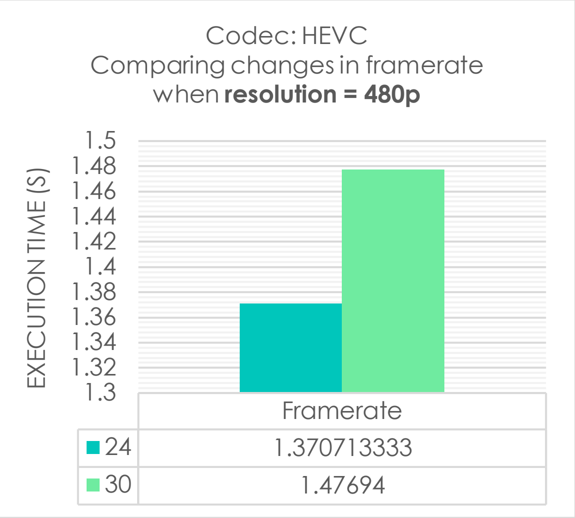
CC-XLarge: Resolution as the constant

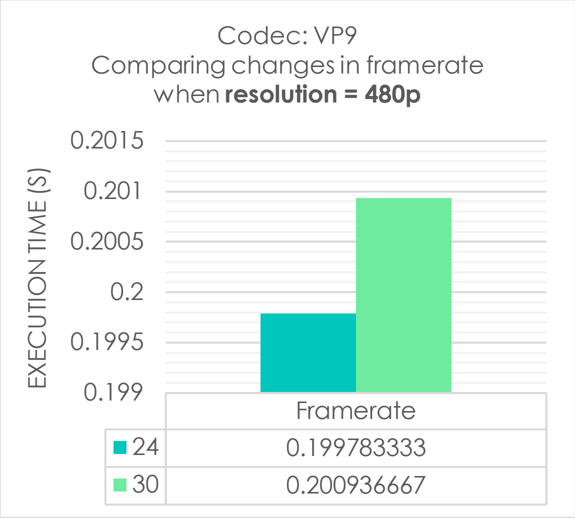


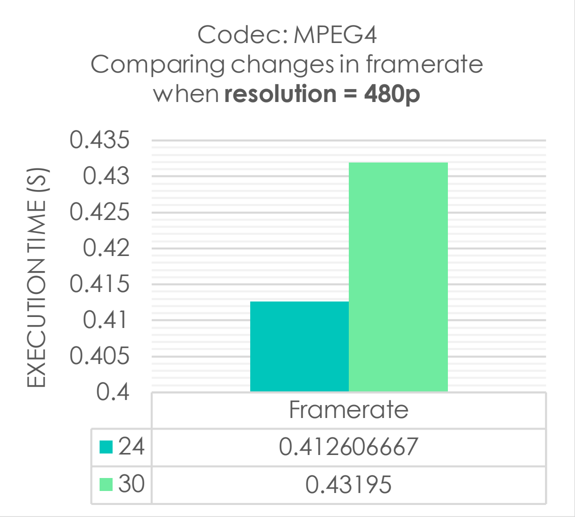


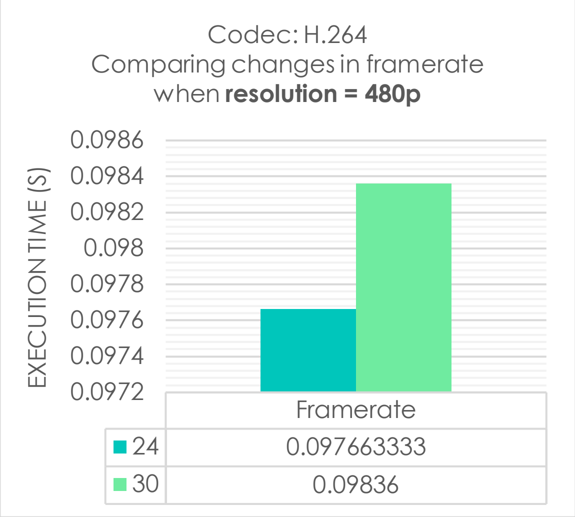


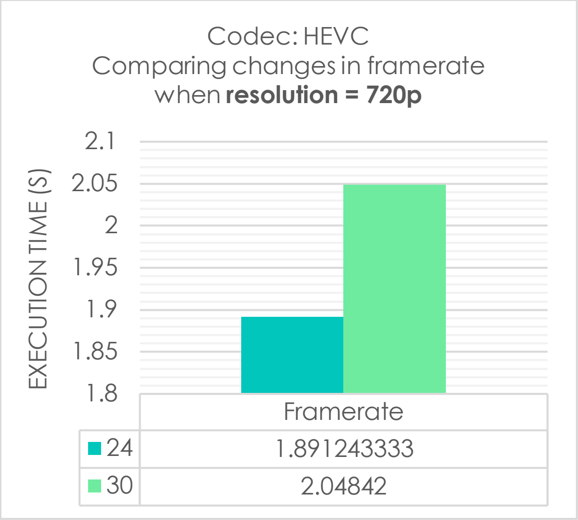




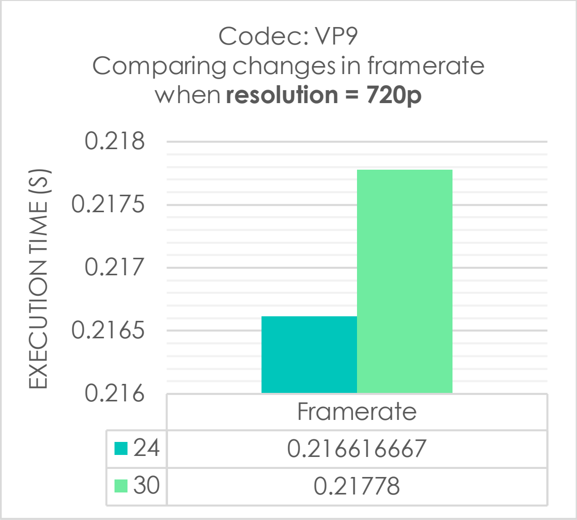




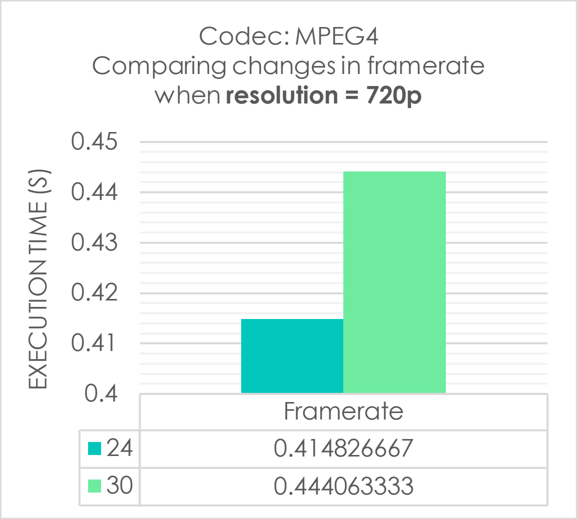




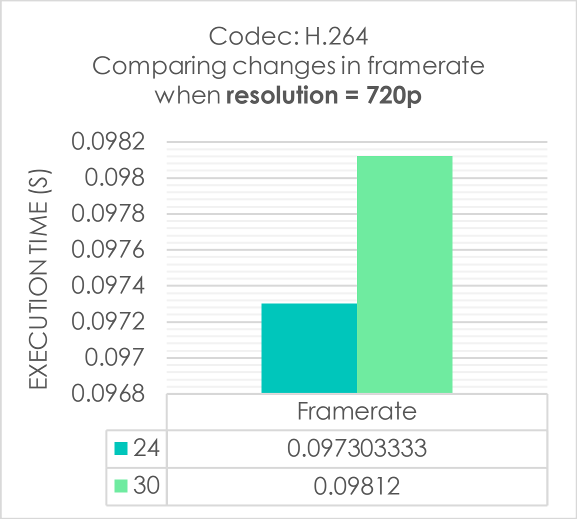
	Framerate
🟦 24	1.891243333
🟩 30	2.04842



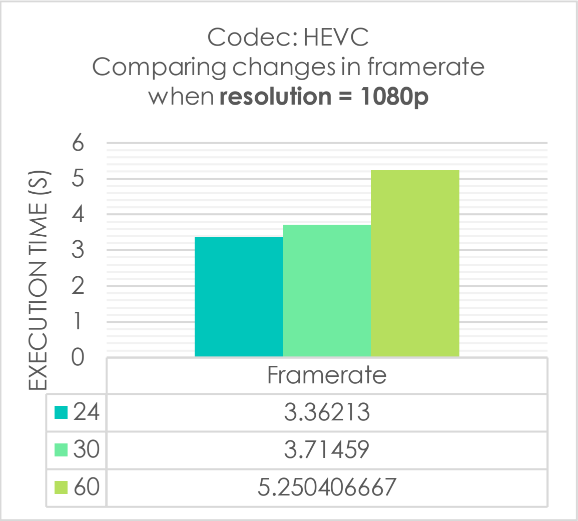
	Framerate
🟦 24	0.216616667
🟩 30	0.21778



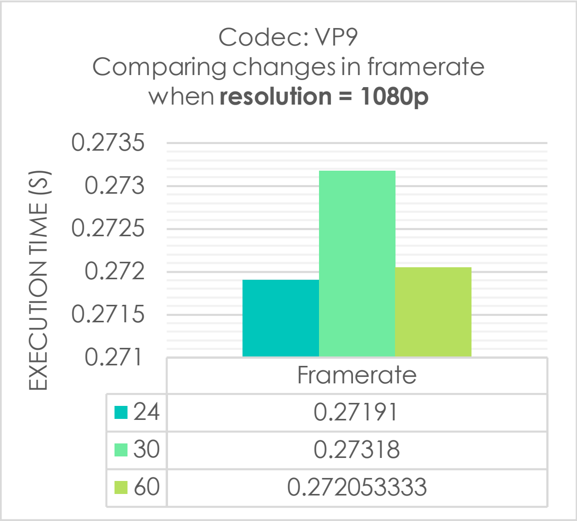
	Framerate
🟦 24	0.414826667
🟩 30	0.444063333



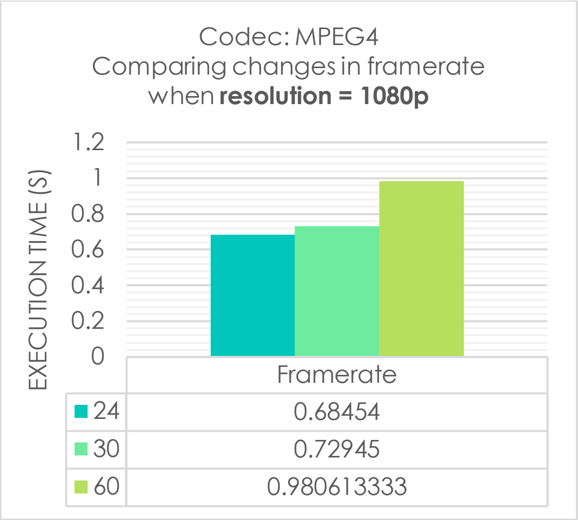
	Framerate
🟦 24	0.097303333
🟩 30	0.09812



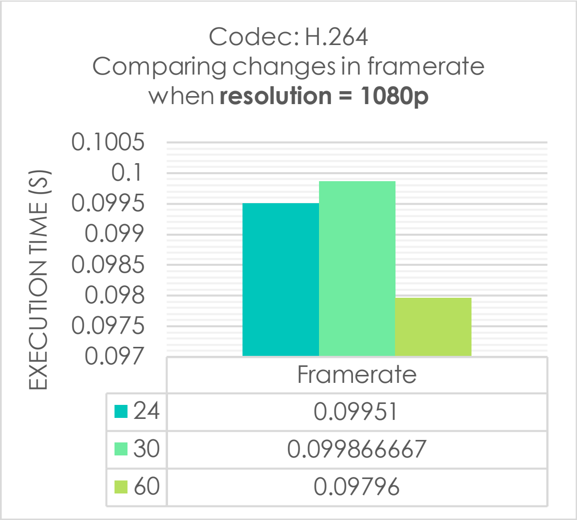
	Framerate
🟦 24	3.36213
🟩 30	3.71459
🟩 60	5.250406667



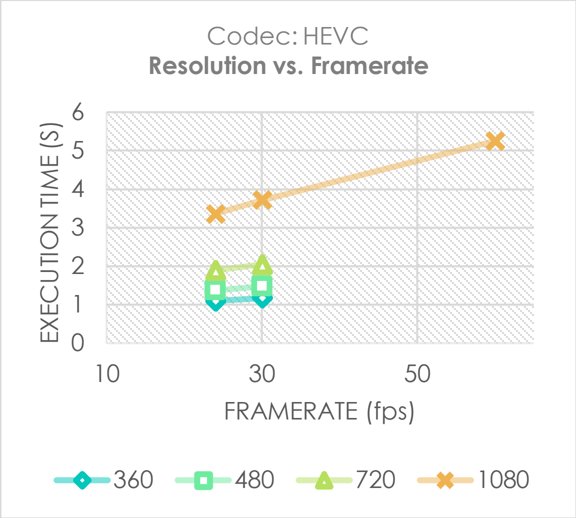
	Framerate
🟦 24	0.27191
🟩 30	0.27318
🟩 60	0.272053333

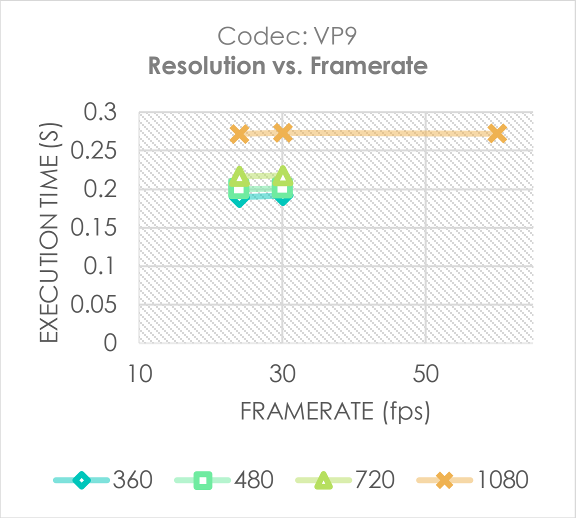


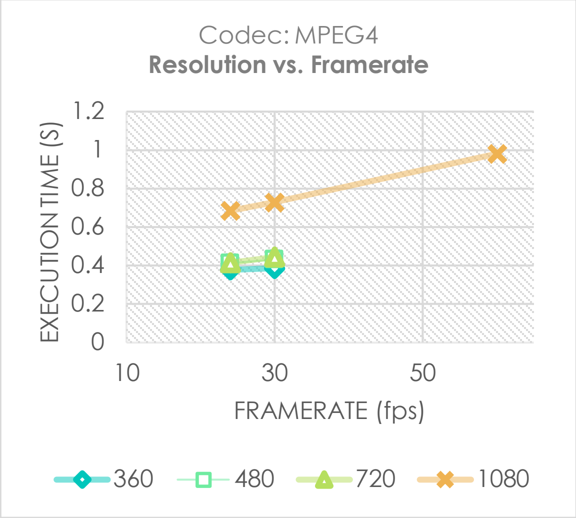
	Framerate
🟦 24	0.68454
🟩 30	0.72945
🟩 60	0.980613333

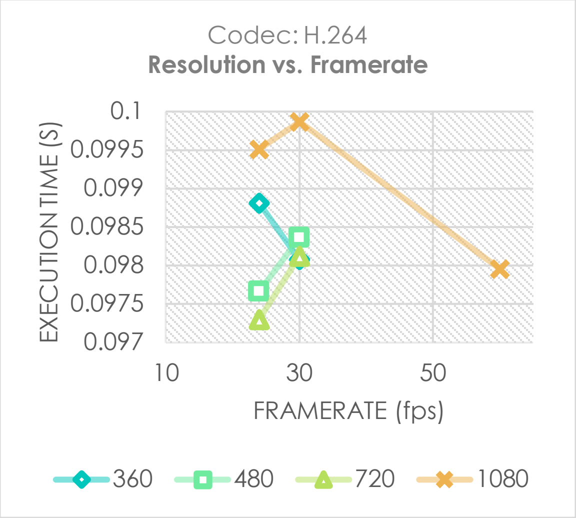


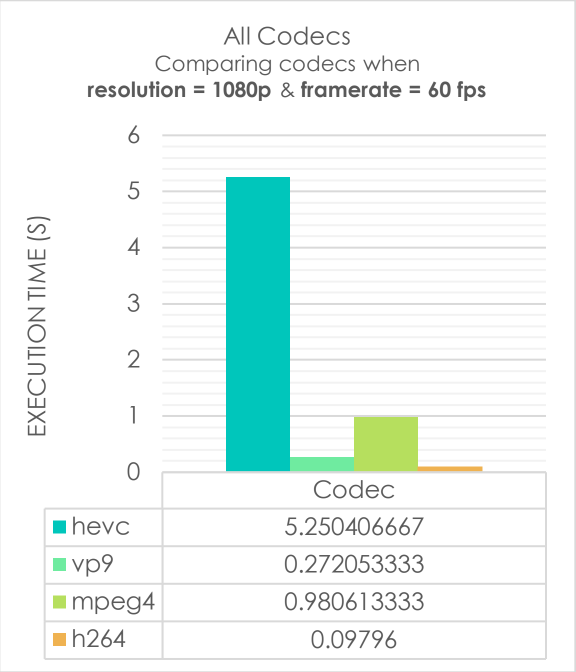
	Framerate
🟦 24	0.09951
🟩 30	0.099866667
🟩 60	0.09796



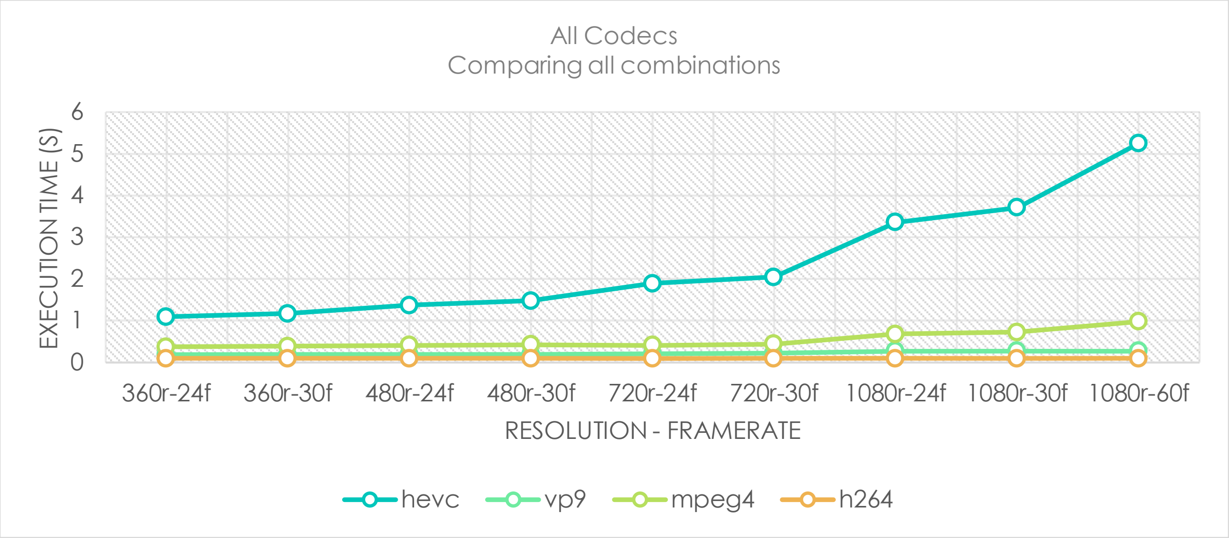




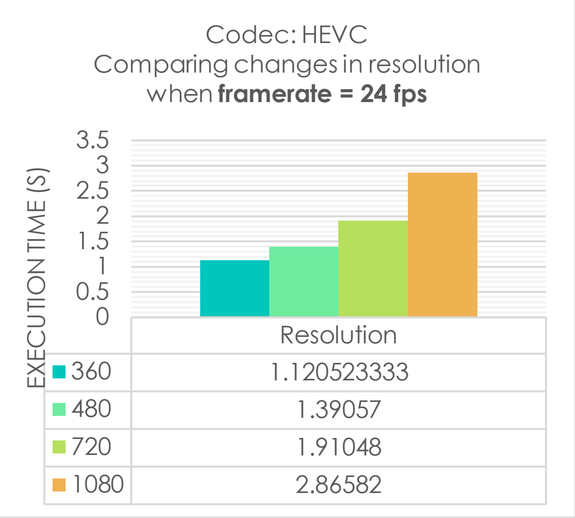




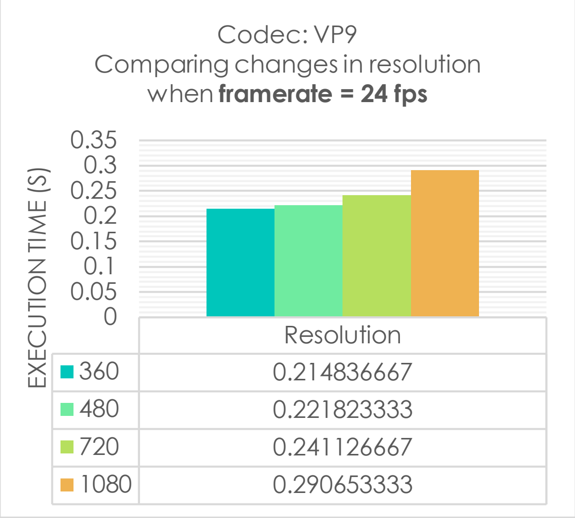
	Codec
hevc	5.250406667
vp9	0.272053333
mpeg4	0.980613333
h264	0.09796



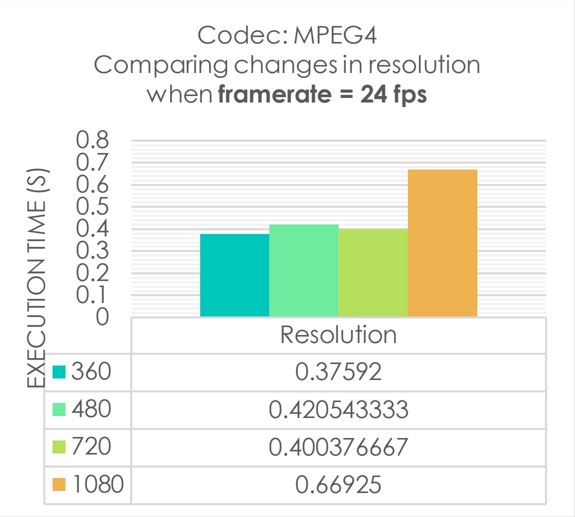
CC-2xLarge: Framerate as the constant


Codec: HEVC
Comparing changes in resolution
when **framerate = 24 fps**

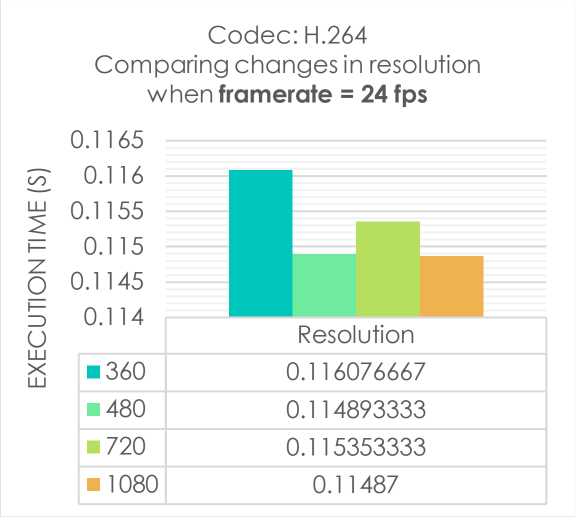
	Resolution
🟦 360	1.120523333
🟩 480	1.39057
🟩 720	1.91048
🟧 1080	2.86582


Codec: VP9
Comparing changes in resolution
when **framerate = 24 fps**

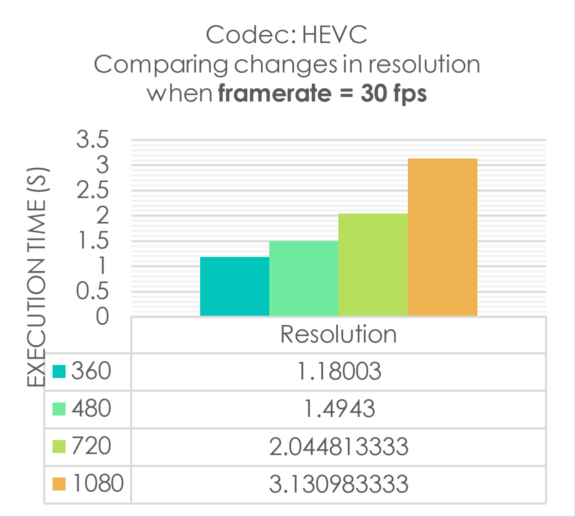
	Resolution
🟦 360	0.214836667
🟩 480	0.221823333
🟩 720	0.241126667
🟧 1080	0.290653333


Codec: MPEG4
Comparing changes in resolution
when **framerate = 24 fps**

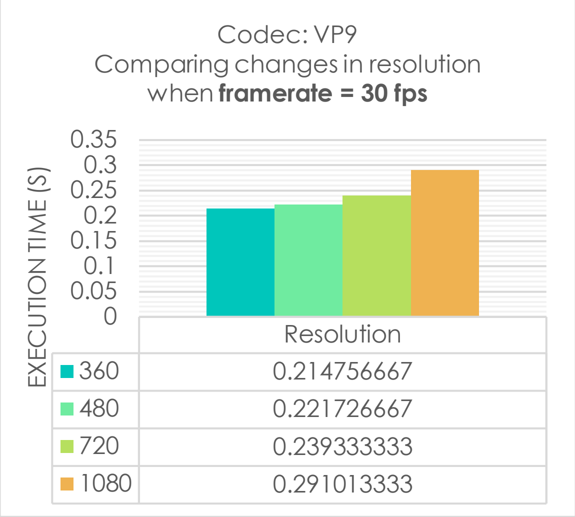
	Resolution
🟦 360	0.37592
🟩 480	0.420543333
🟩 720	0.400376667
🟧 1080	0.66925


Codec: H.264
Comparing changes in resolution
when **framerate = 24 fps**

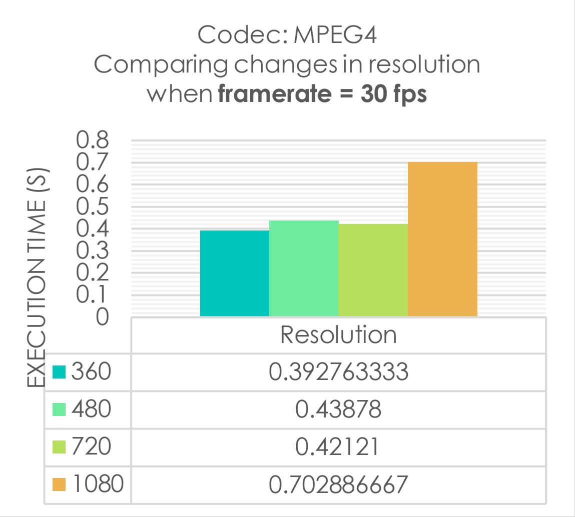
	Resolution
🟦 360	0.116076667
🟩 480	0.114893333
🟩 720	0.115353333
🟧 1080	0.11487


Codec: HEVC
Comparing changes in resolution
when **framerate = 30 fps**

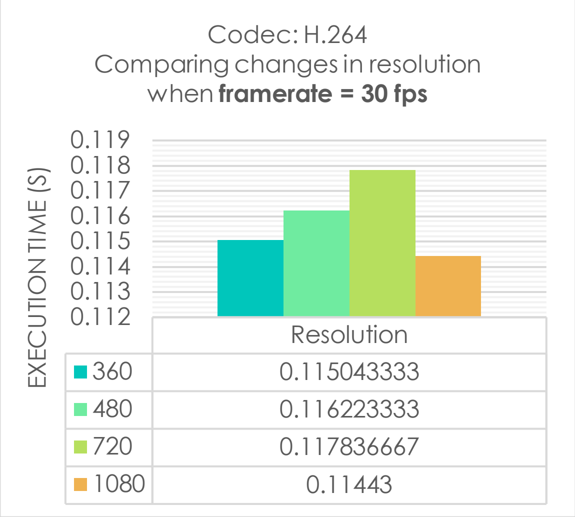
	Resolution
🟦 360	1.18003
🟩 480	1.4943
🟩 720	2.044813333
🟧 1080	3.130983333


Codec: VP9
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
🟦 360	0.214756667
🟩 480	0.221726667
🟩 720	0.239333333
🟧 1080	0.291013333

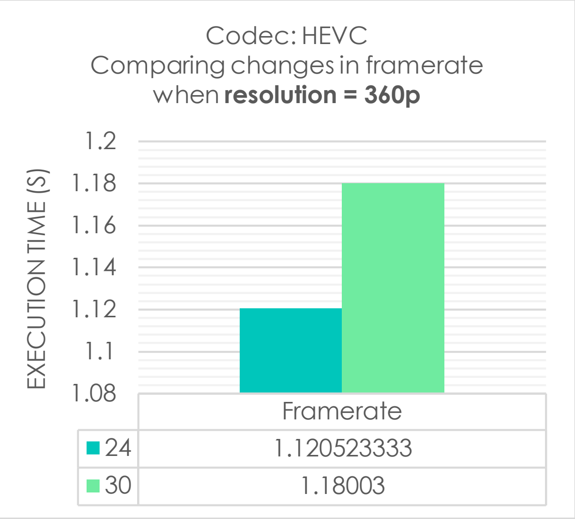

Codec: MPEG4
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
🟦 360	0.392763333
🟩 480	0.43878
🟩 720	0.42121
🟧 1080	0.702886667

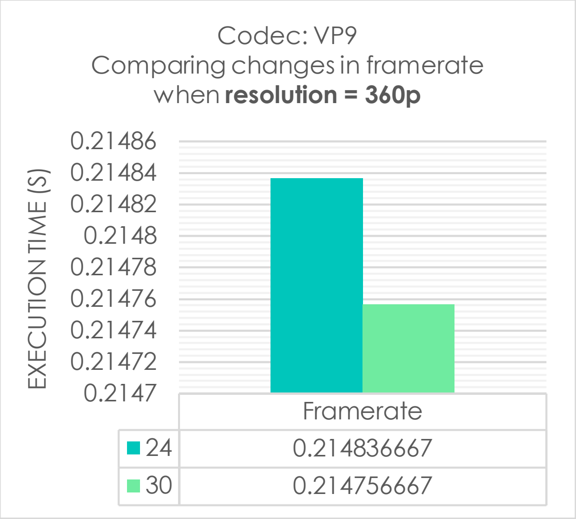

Codec: H.264
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
🟦 360	0.115043333
🟩 480	0.116223333
🟩 720	0.117836667
🟧 1080	0.11443

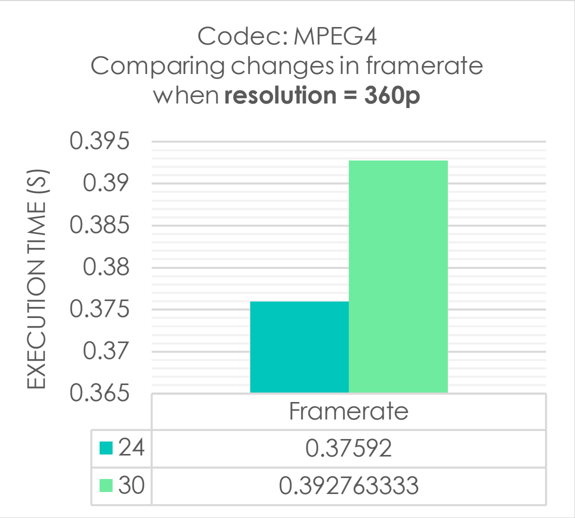
CC-2xLarge: Resolution as the constant



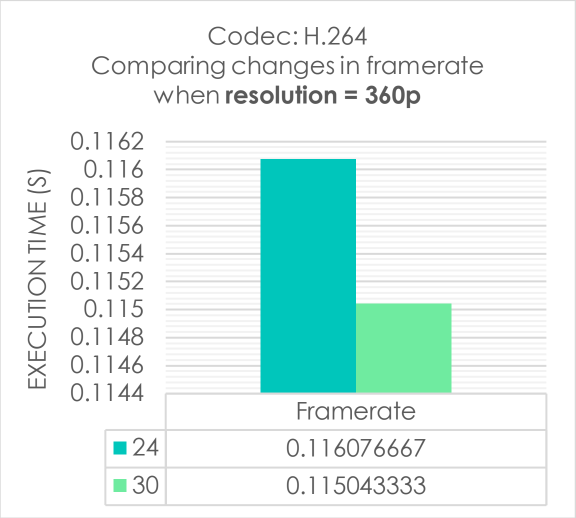
	Framerate
■ 24	1.120523333
■ 30	1.18003



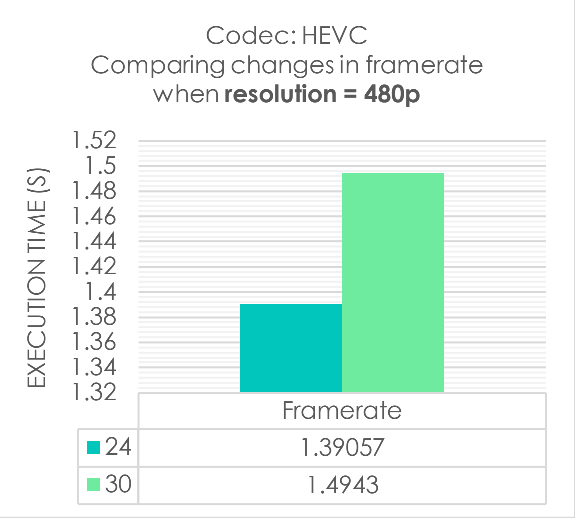
	Framerate
■ 24	0.214836667
■ 30	0.214756667



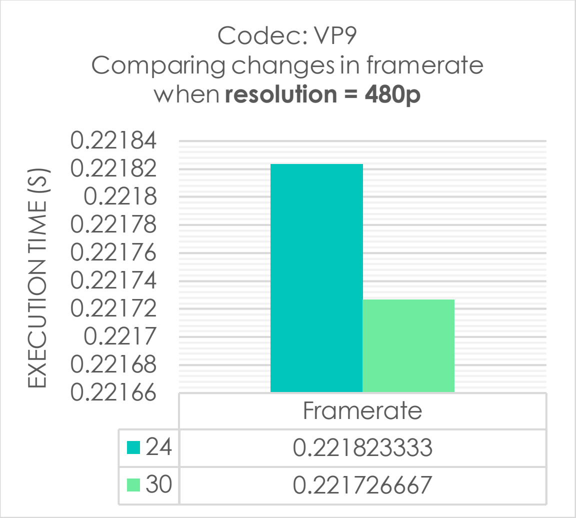
	Framerate
■ 24	0.37592
■ 30	0.392763333



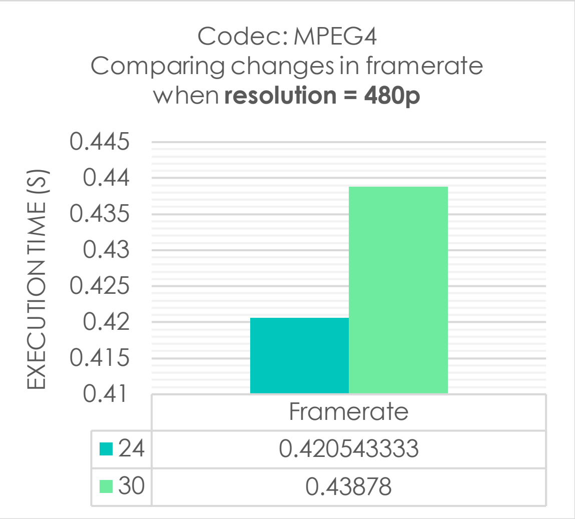
	Framerate
■ 24	0.116076667
■ 30	0.115043333



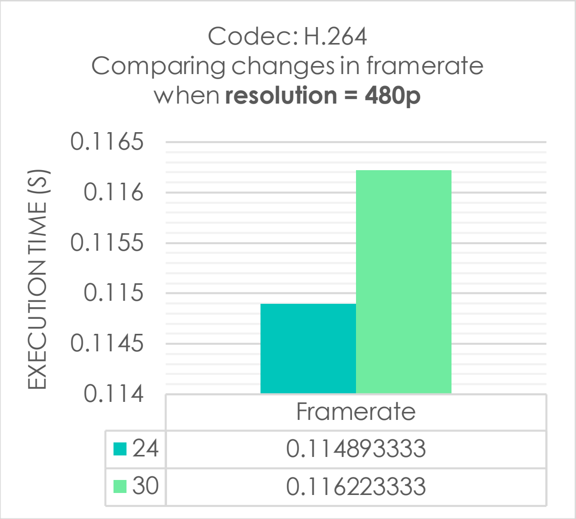
	Framerate
■ 24	1.39057
■ 30	1.4943



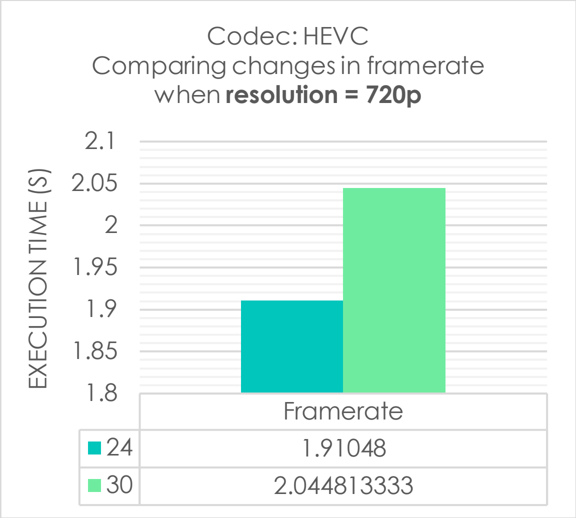
	Framerate
■ 24	0.221823333
■ 30	0.221726667

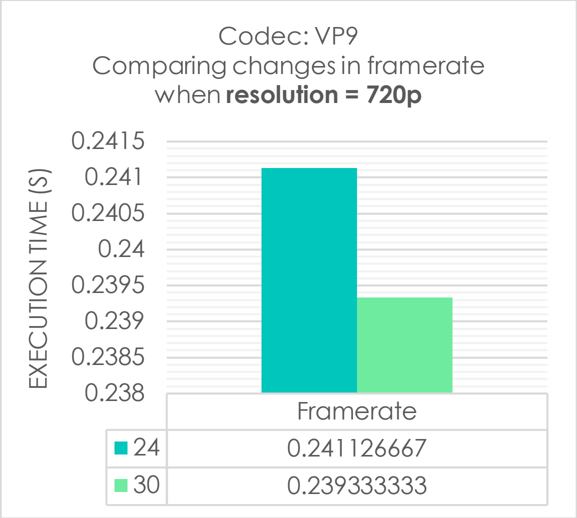


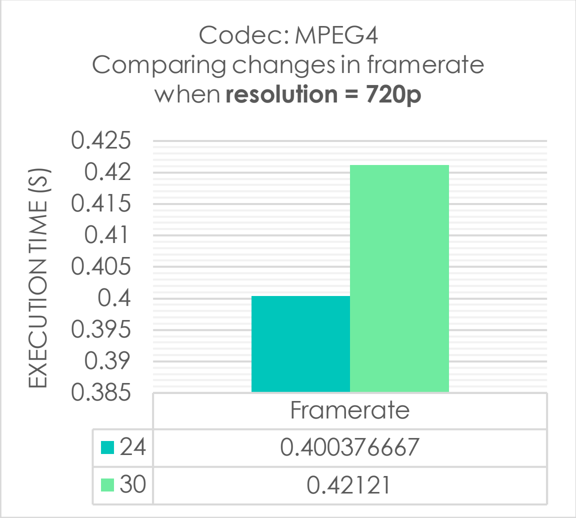
	Framerate
■ 24	0.420543333
■ 30	0.43878

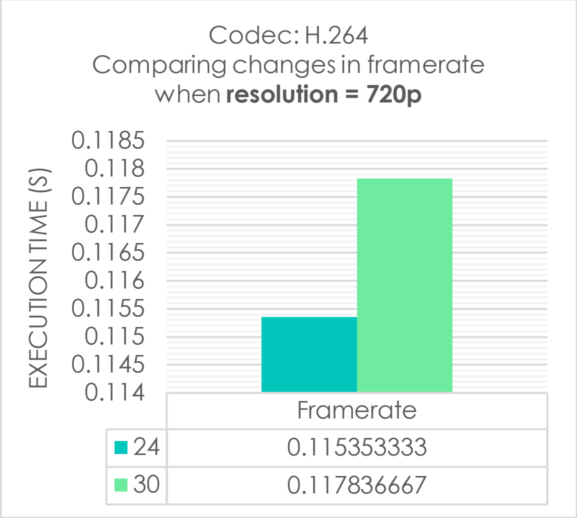


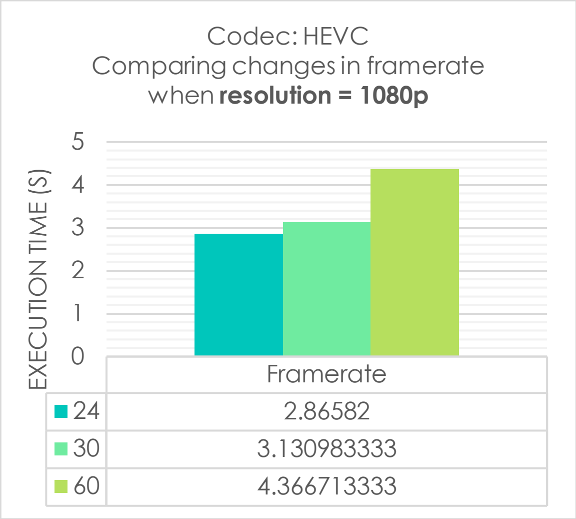
	Framerate
■ 24	0.114893333
■ 30	0.116223333

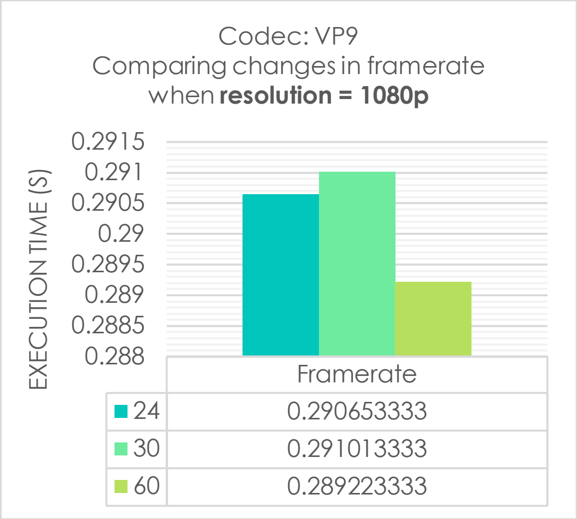

Codec: HEVC
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
2.1
2.05
2
1.95
1.9
1.85
1.8
Framerate
24
1.91048
30
2.044813333

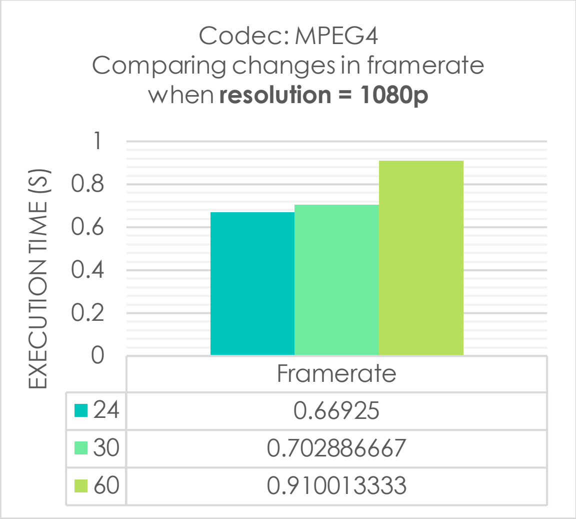

Codec: VP9
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.2415
0.241
0.2405
0.24
0.2395
0.239
0.2385
0.238
Framerate
24
0.241126667
30
0.239333333

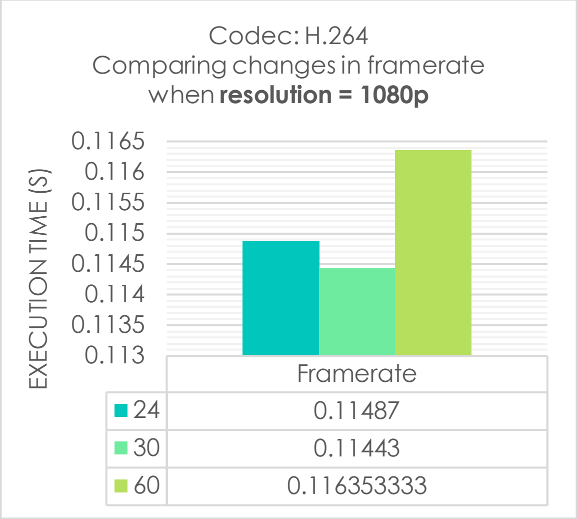

Codec: MPEG4
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.425
0.42
0.415
0.41
0.405
0.4
0.395
0.39
0.385
Framerate
24
0.400376667
30
0.42121

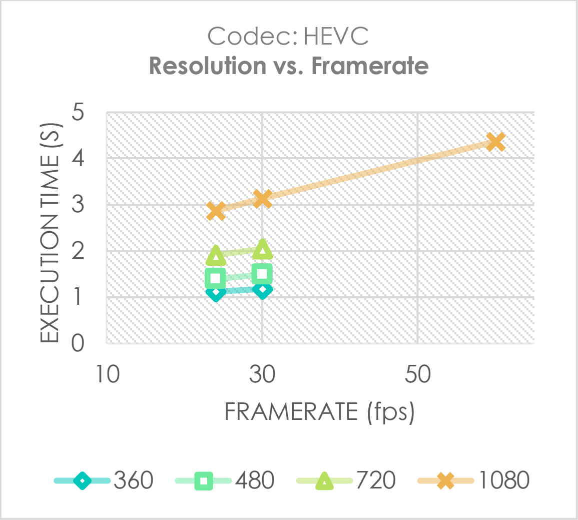

Codec: H.264
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.1185
0.118
0.1175
0.117
0.1165
0.116
0.1155
0.115
0.1145
0.114
Framerate
24
0.115353333
30
0.117836667

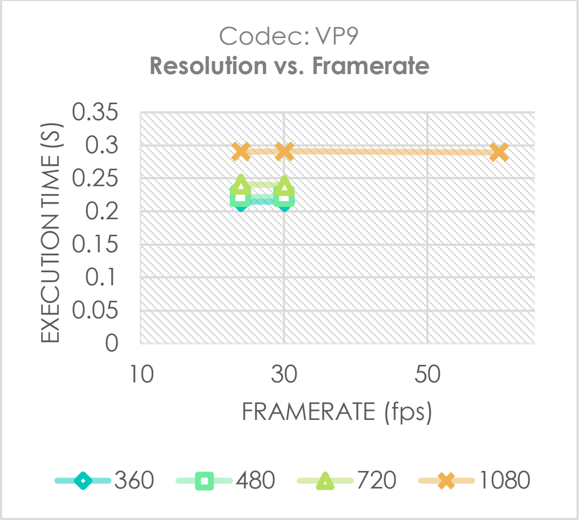

Codec: HEVC
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
5
4
3
2
1
0
Framerate
24
2.86582
30
3.130983333
60
4.366713333

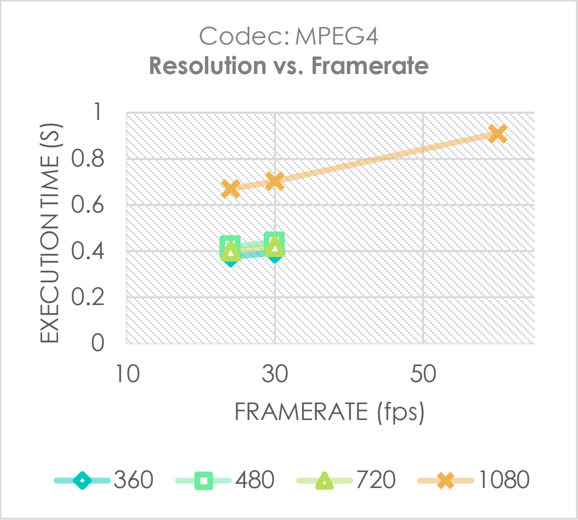

Codec: VP9
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.2915
0.291
0.2905
0.29
0.2895
0.289
0.2885
0.288
Framerate
24
0.290653333
30
0.291013333
60
0.289223333

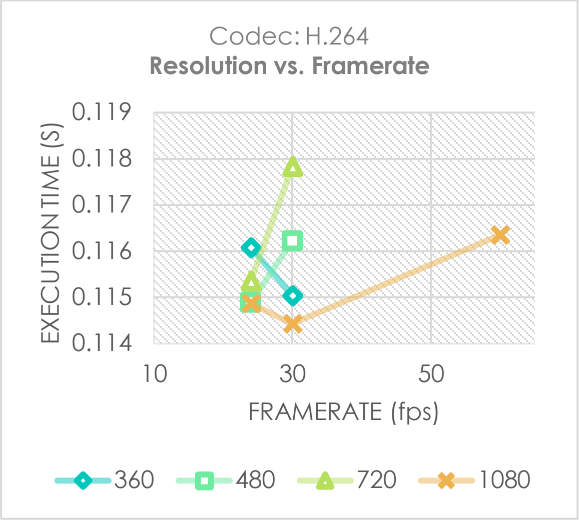

Codec: MPEG4
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
1
0.8
0.6
0.4
0.2
0
Framerate
24
0.66925
30
0.702886667
60
0.910013333

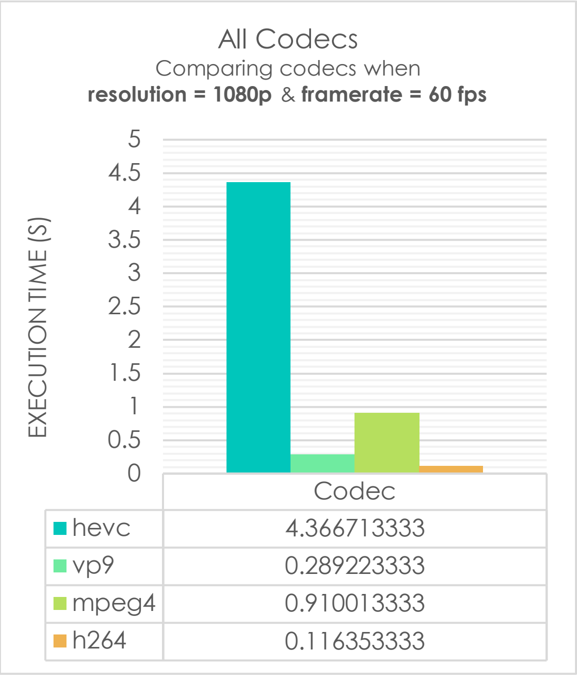

Codec: H.264
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.1165
0.116
0.1155
0.115
0.1145
0.114
0.1135
0.113
Framerate
24
0.11487
30
0.11443
60
0.116353333



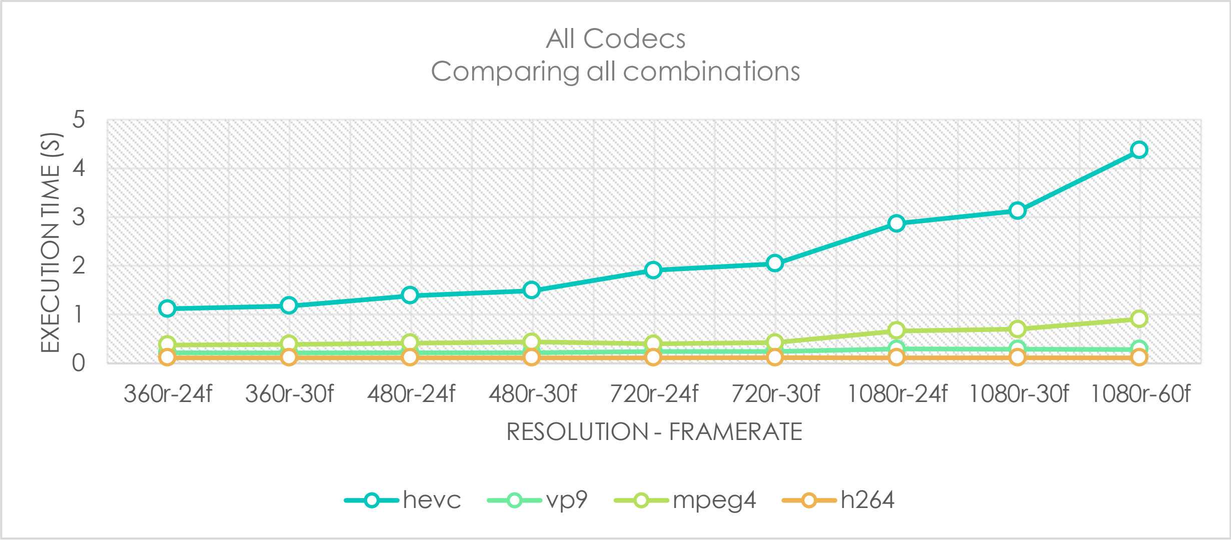




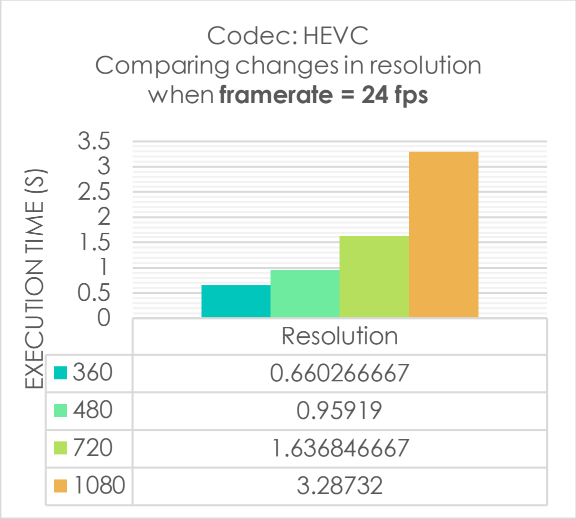




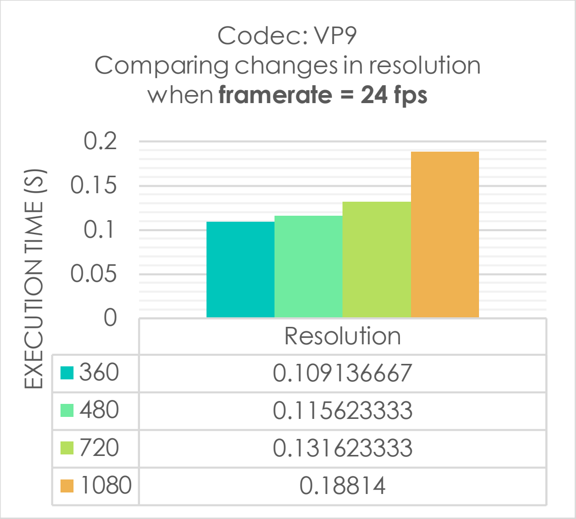
	Codec
hevc	4.366713333
vp9	0.289223333
mpeg4	0.910013333
h264	0.116353333



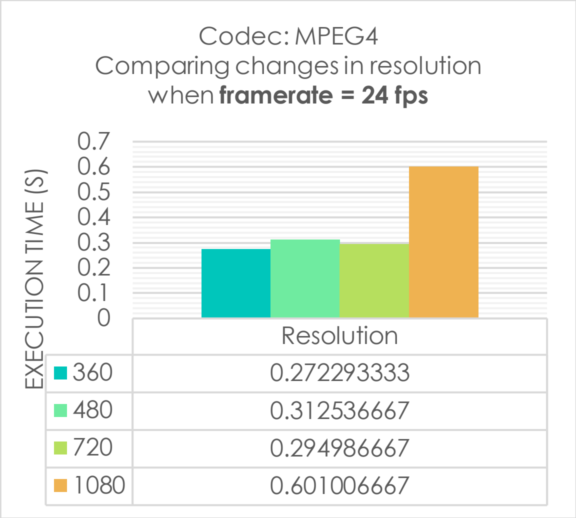
AWS-CO: Framerate as the constant


Codec: HEVC
Comparing changes in resolution
when **framerate = 24 fps**

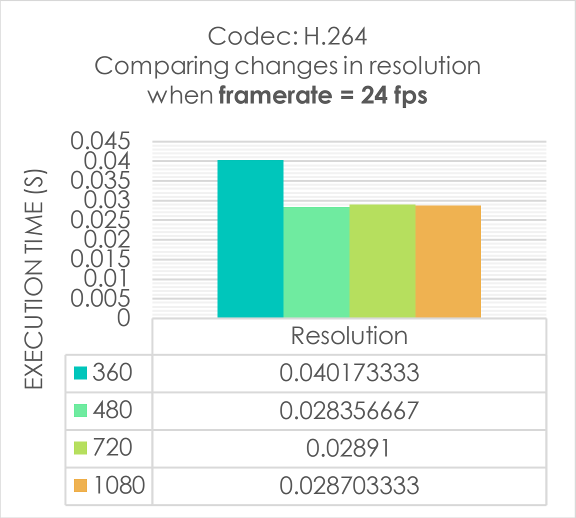
	Resolution
🟦 360	0.660266667
🟩 480	0.95919
🟩 720	1.636846667
🟧 1080	3.28732


Codec: VP9
Comparing changes in resolution
when **framerate = 24 fps**

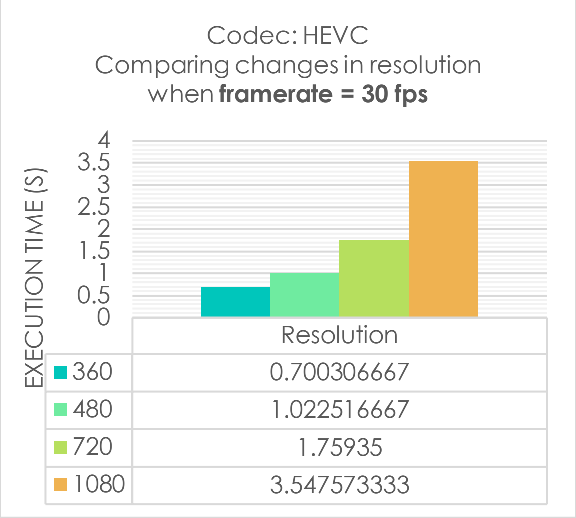
	Resolution
🟦 360	0.109136667
🟩 480	0.115623333
🟩 720	0.131623333
🟧 1080	0.18814


Codec: MPEG4
Comparing changes in resolution
when **framerate = 24 fps**

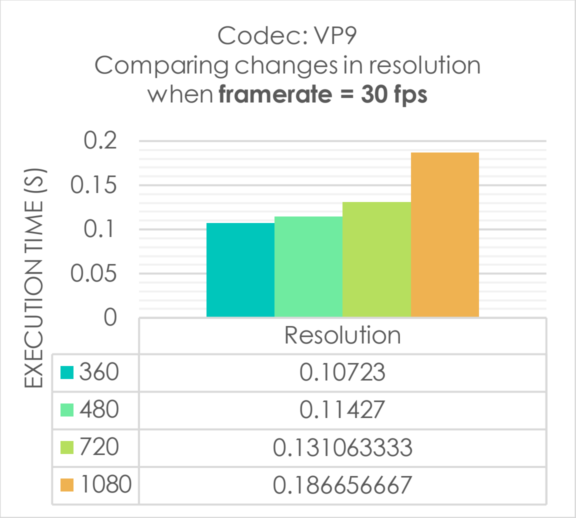
	Resolution
🟦 360	0.272293333
🟩 480	0.312536667
🟩 720	0.294986667
🟧 1080	0.601006667


Codec: H.264
Comparing changes in resolution
when **framerate = 24 fps**

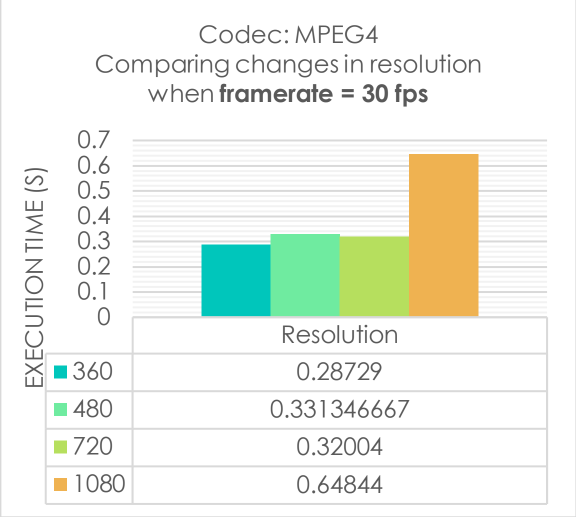
	Resolution
🟦 360	0.040173333
🟩 480	0.028356667
🟩 720	0.02891
🟧 1080	0.028703333


Codec: HEVC
Comparing changes in resolution
when **framerate = 30 fps**

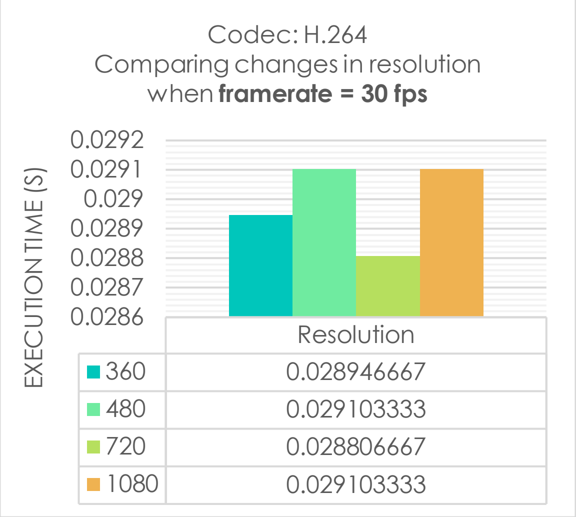
	Resolution
🟦 360	0.700306667
🟩 480	1.022516667
🟩 720	1.75935
🟧 1080	3.547573333


Codec: VP9
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
🟦 360	0.10723
🟩 480	0.11427
🟩 720	0.131063333
🟧 1080	0.186656667

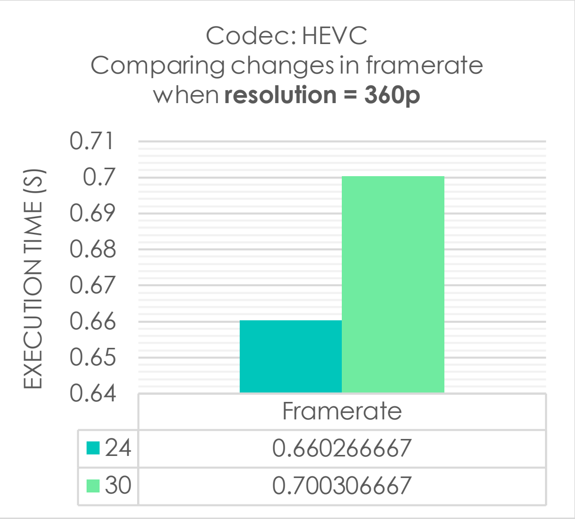

Codec: MPEG4
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
🟦 360	0.28729
🟩 480	0.331346667
🟩 720	0.32004
🟧 1080	0.64844

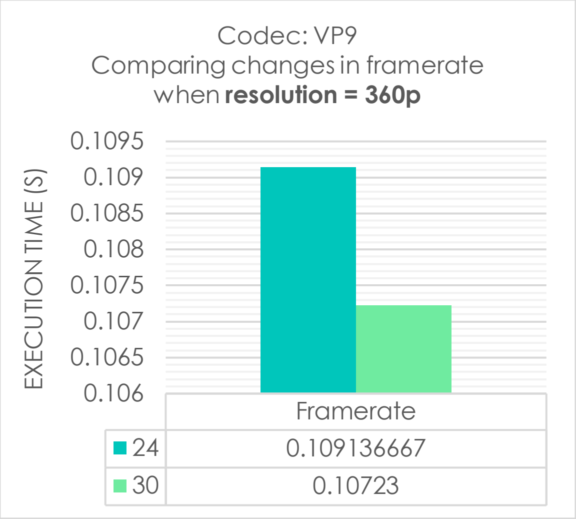

Codec: H.264
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
🟦 360	0.028946667
🟩 480	0.029103333
🟩 720	0.028806667
🟧 1080	0.029103333

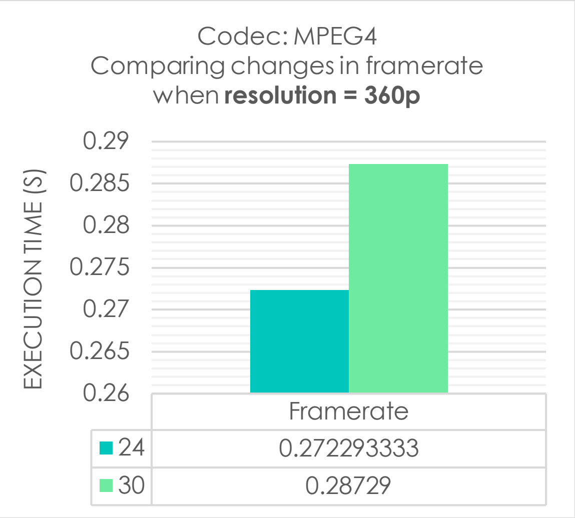
AWS-CO: Resolution as the constant


Codec: HEVC
Comparing changes in framerate
when **resolution = 360p**

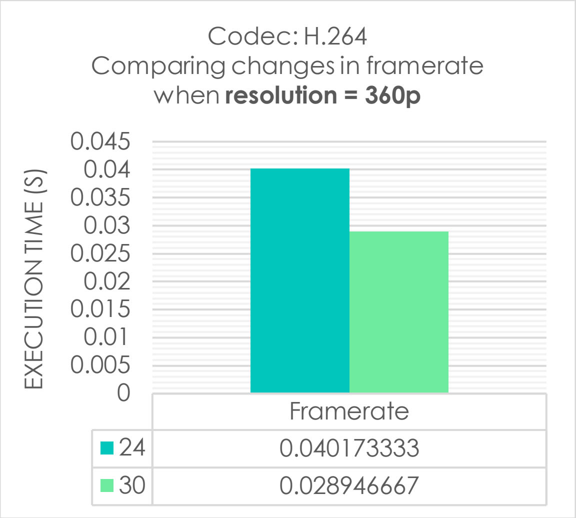
	Framerate
■ 24	0.660266667
■ 30	0.700306667


Codec: VP9
Comparing changes in framerate
when **resolution = 360p**

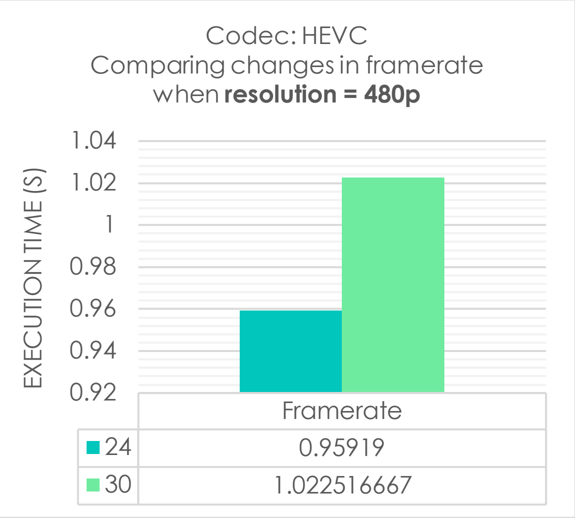
	Framerate
■ 24	0.109136667
■ 30	0.10723


Codec: MPEG4
Comparing changes in framerate
when **resolution = 360p**

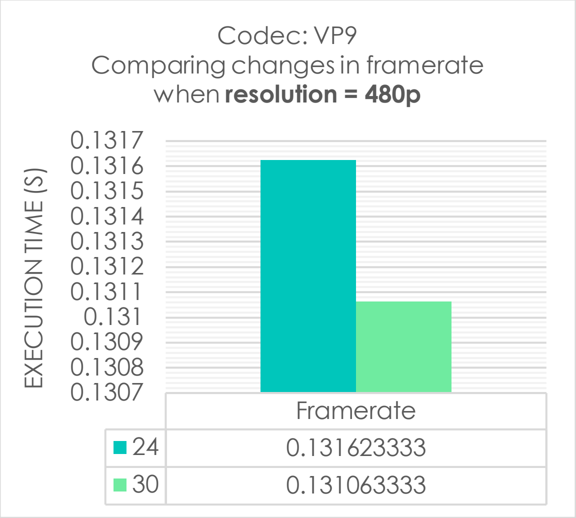
	Framerate
■ 24	0.272293333
■ 30	0.28729


Codec: H.264
Comparing changes in framerate
when **resolution = 360p**

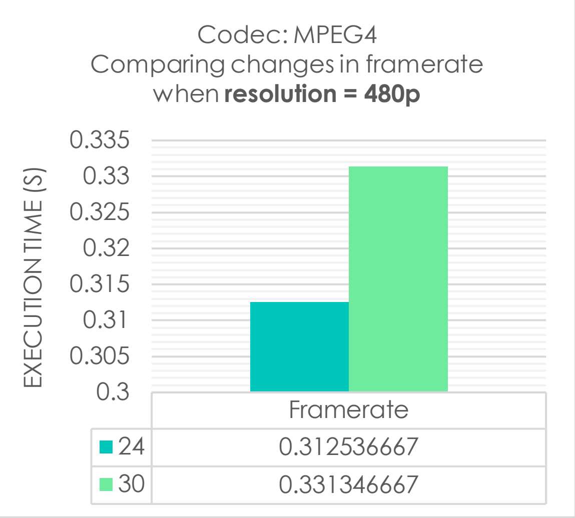
	Framerate
■ 24	0.040173333
■ 30	0.028946667


Codec: HEVC
Comparing changes in framerate
when **resolution = 480p**

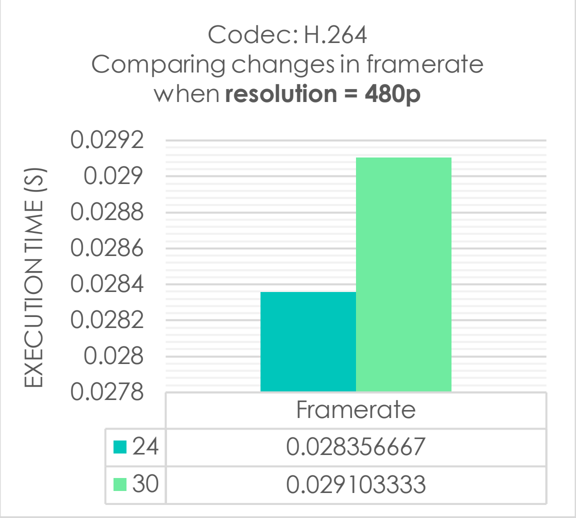
	Framerate
■ 24	0.95919
■ 30	1.022516667


Codec: VP9
Comparing changes in framerate
when **resolution = 480p**

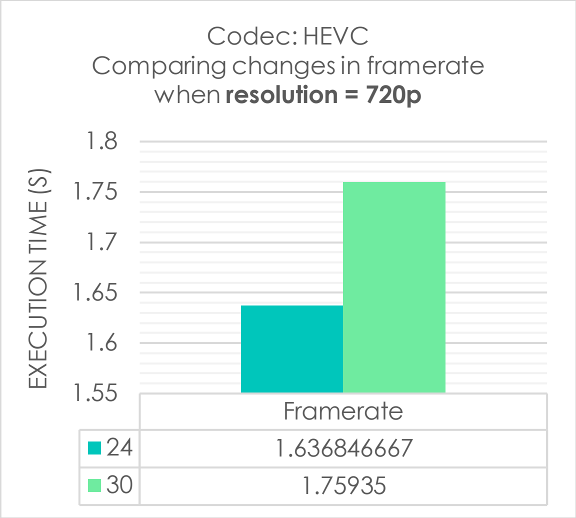
	Framerate
■ 24	0.131623333
■ 30	0.131063333

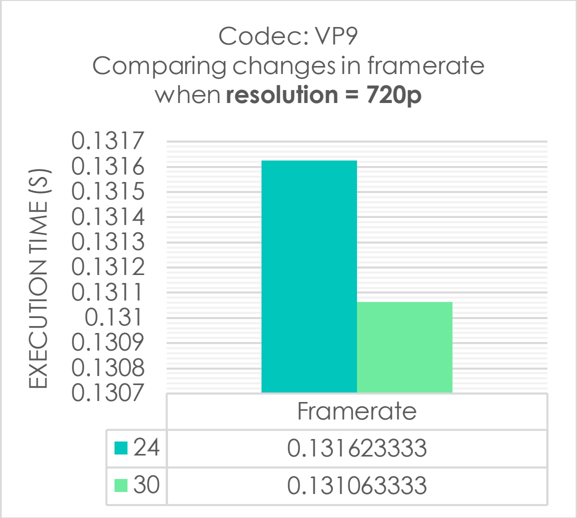

Codec: MPEG4
Comparing changes in framerate
when **resolution = 480p**

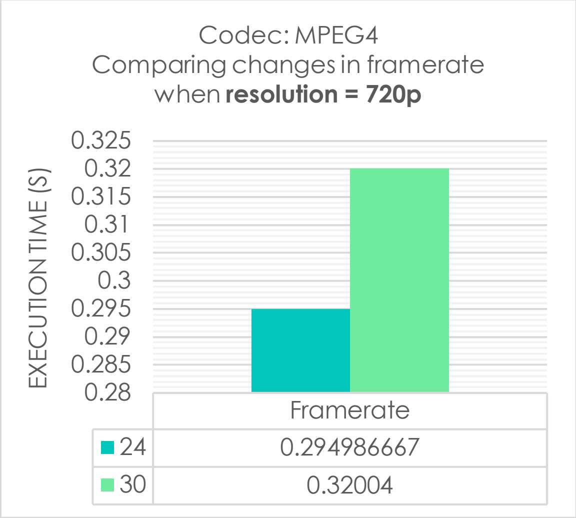
	Framerate
■ 24	0.312536667
■ 30	0.331346667

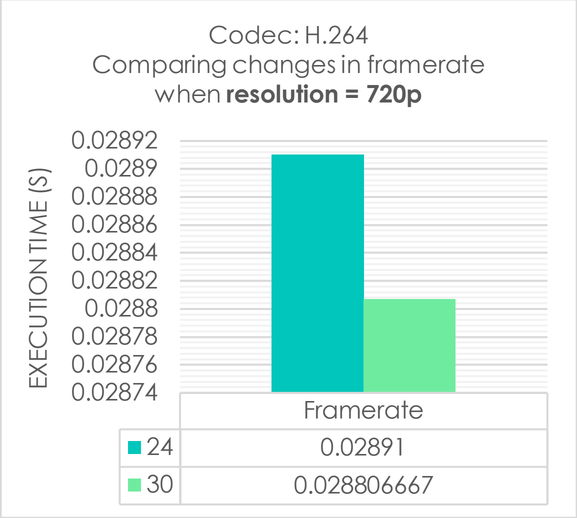

Codec: H.264
Comparing changes in framerate
when **resolution = 480p**

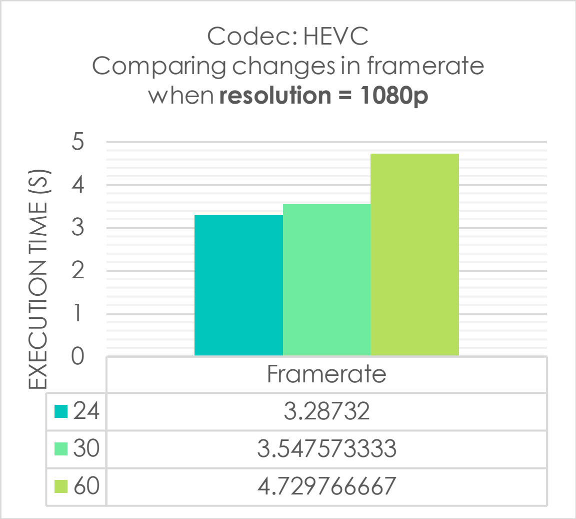
	Framerate
■ 24	0.028356667
■ 30	0.029103333

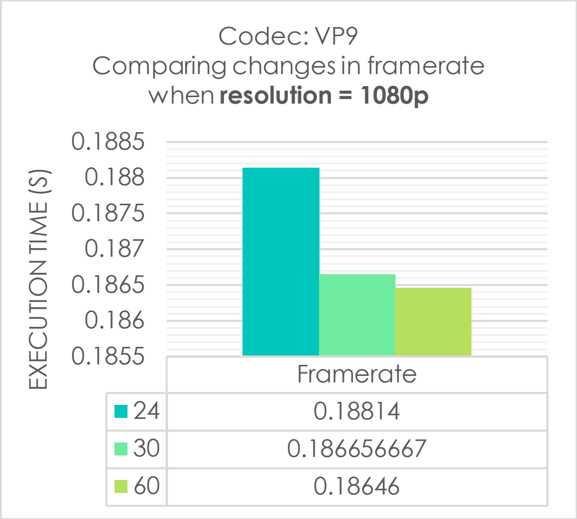

Codec: HEVC
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
1.8
1.75
1.7
1.65
1.6
1.55
Framerate
24 1.636846667
30 1.75935

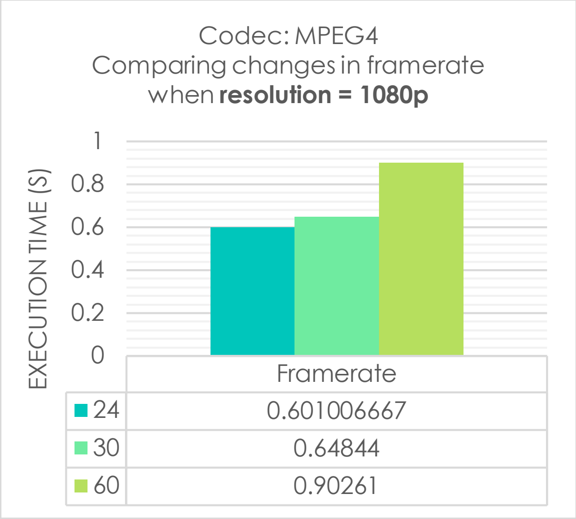

Codec: VP9
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.1317
0.1316
0.1315
0.1314
0.1313
0.1312
0.1311
0.131
0.1309
0.1308
0.1307
Framerate
24 0.131623333
30 0.131063333

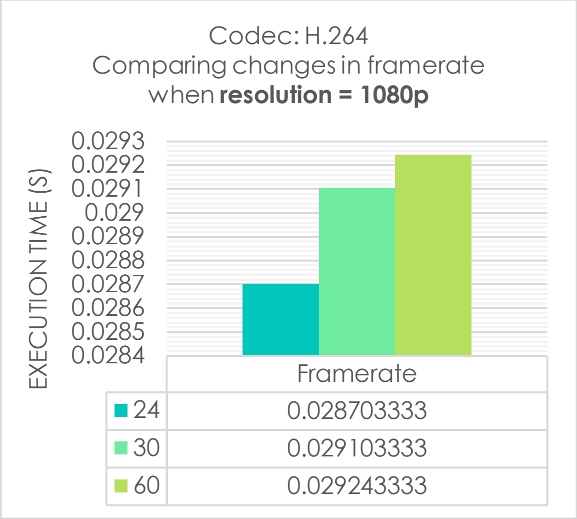

Codec: MPEG4
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.325
0.32
0.315
0.31
0.305
0.3
0.295
0.29
0.285
0.28
Framerate
24 0.294986667
30 0.32004

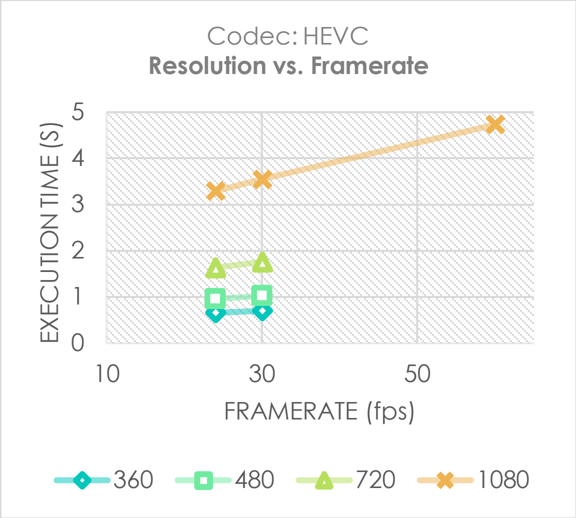

Codec: H.264
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.02892
0.0289
0.02888
0.02886
0.02884
0.02882
0.0288
0.02878
0.02876
0.02874
Framerate
24 0.02891
30 0.028806667

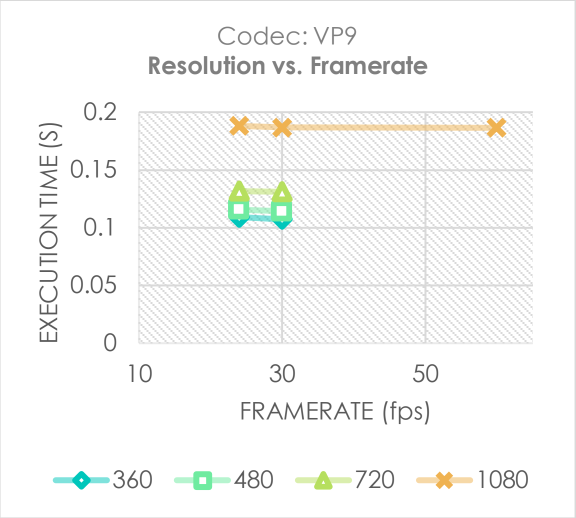

Codec: HEVC
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
5
4
3
2
1
0
Framerate
24 3.28732
30 3.547573333
60 4.729766667

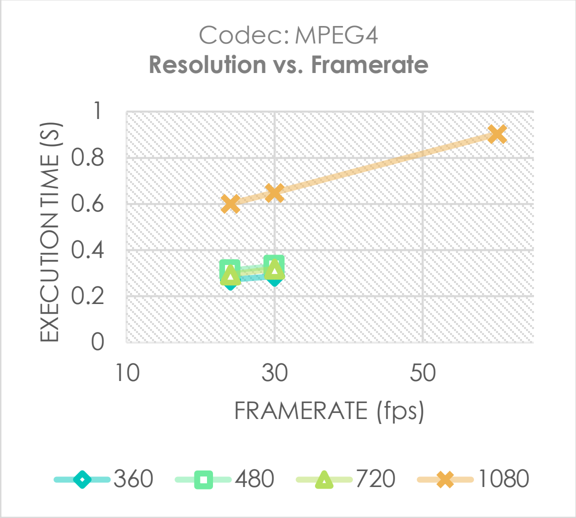

Codec: VP9
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.1885
0.188
0.1875
0.187
0.1865
0.186
0.1855
Framerate
24 0.18814
30 0.186656667
60 0.18646

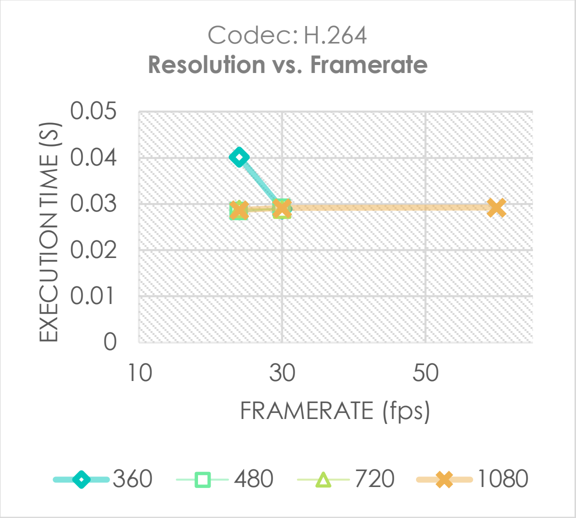

Codec: MPEG4
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
1
0.8
0.6
0.4
0.2
0
Framerate
24 0.601006667
30 0.64844
60 0.90261

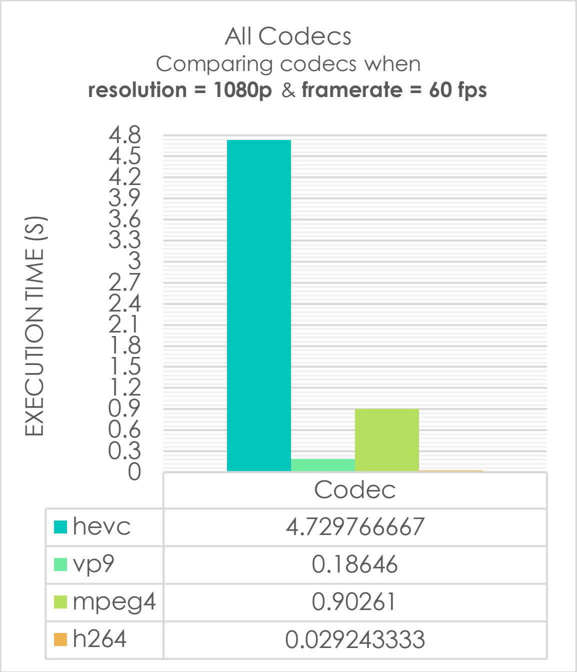

Codec: H.264
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.0293
0.0292
0.0291
0.029
0.0289
0.0288
0.0287
0.0286
0.0285
0.0284
Framerate
24 0.028703333
30 0.029103333
60 0.029243333



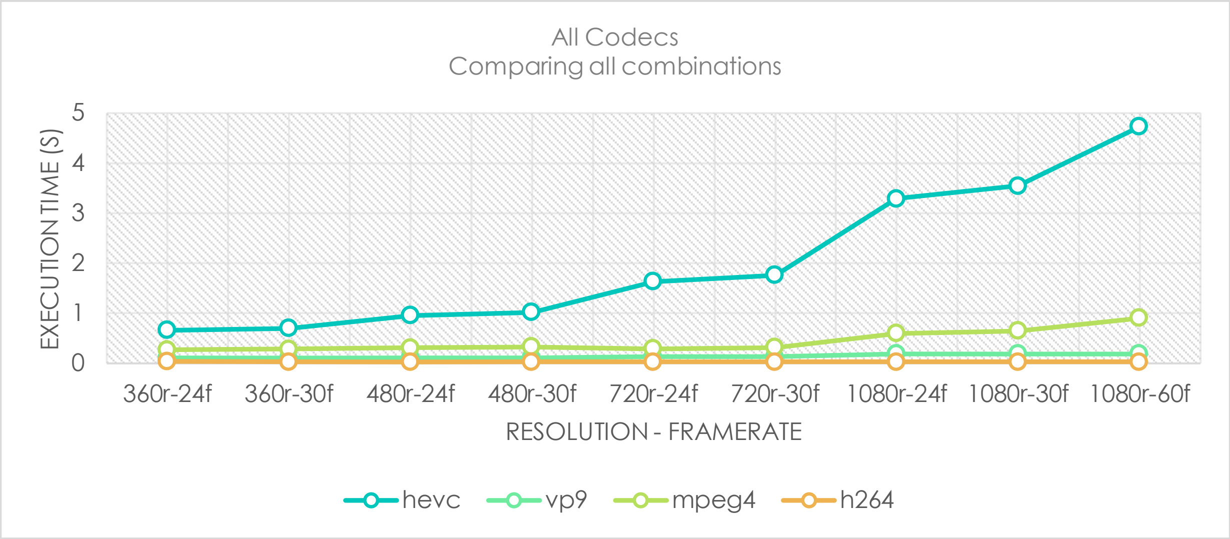




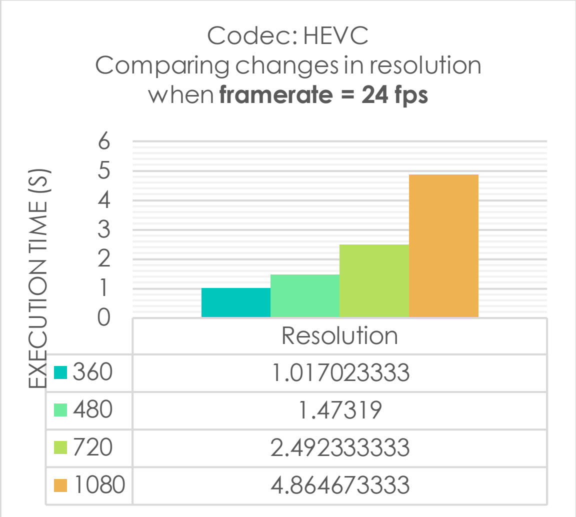




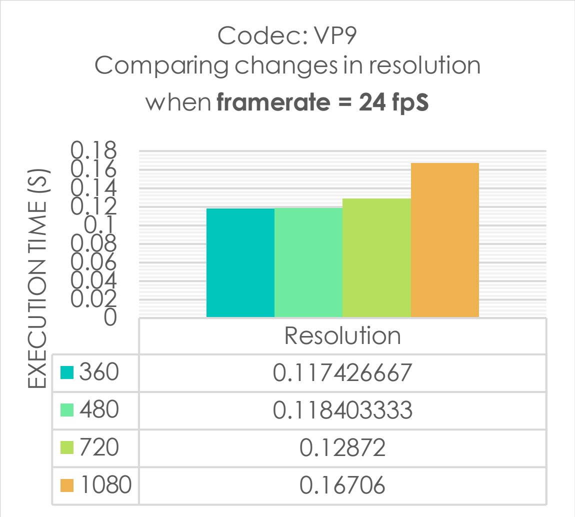
	Codec
hevc	4.729766667
vp9	0.18646
mpeg4	0.90261
h264	0.029243333



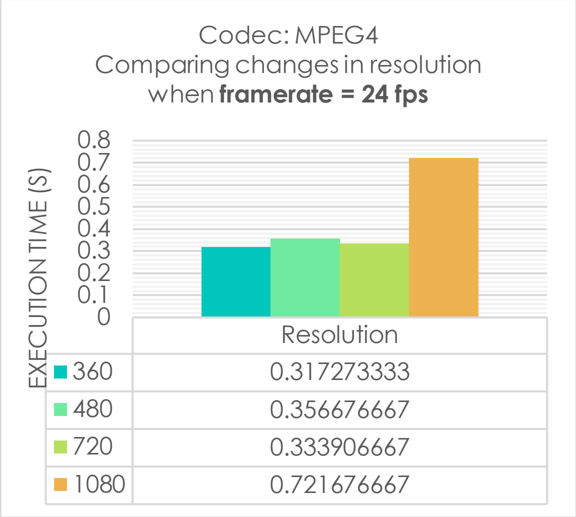
AWS-GPA: Framerate as the constant


Codec: HEVC
Comparing changes in resolution
when **framerate = 24 fps**

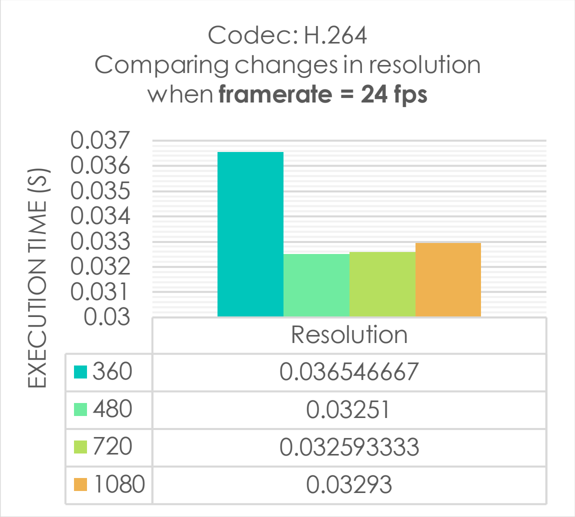
	Resolution
■ 360	1.017023333
■ 480	1.47319
■ 720	2.492333333
■ 1080	4.864673333


Codec: VP9
Comparing changes in resolution
when **framerate = 24 fpS**

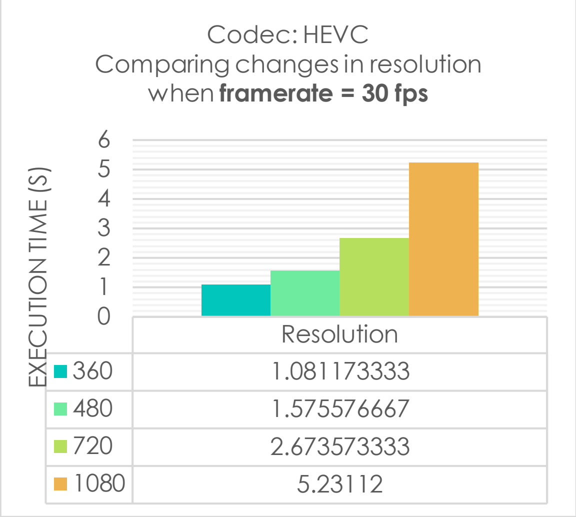
	Resolution
■ 360	0.117426667
■ 480	0.118403333
■ 720	0.12872
■ 1080	0.16706


Codec: MPEG4
Comparing changes in resolution
when **framerate = 24 fps**

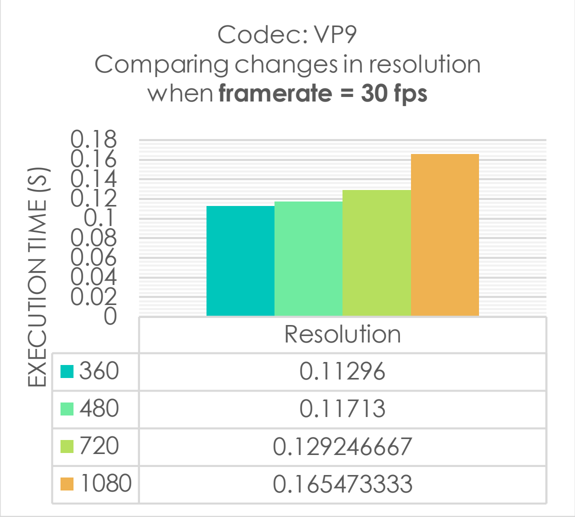
	Resolution
■ 360	0.317273333
■ 480	0.356676667
■ 720	0.333906667
■ 1080	0.721676667


Codec: H.264
Comparing changes in resolution
when **framerate = 24 fps**

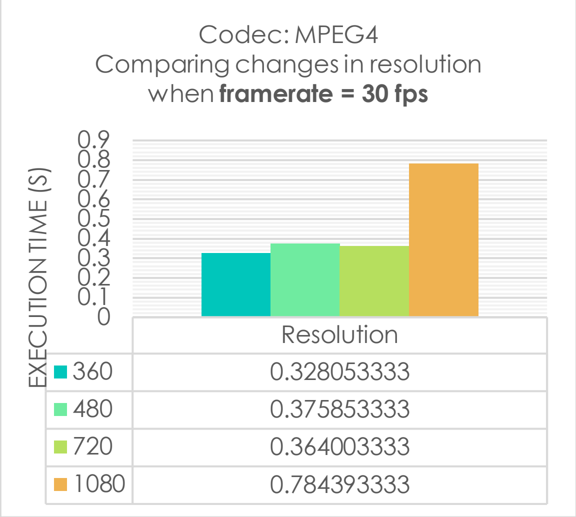
	Resolution
■ 360	0.036546667
■ 480	0.03251
■ 720	0.032593333
■ 1080	0.03293


Codec: HEVC
Comparing changes in resolution
when **framerate = 30 fps**

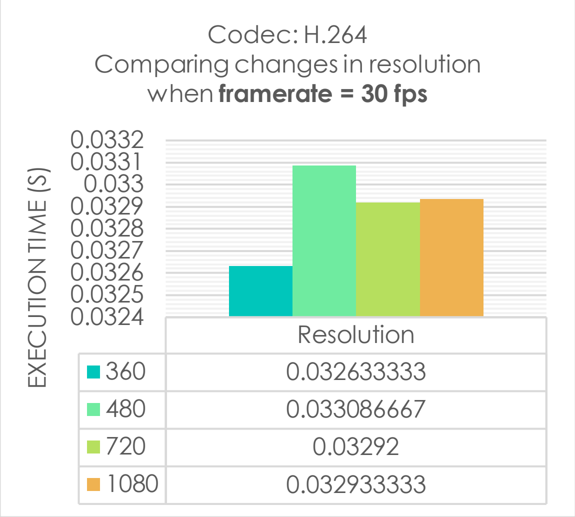
	Resolution
■ 360	1.081173333
■ 480	1.575576667
■ 720	2.673573333
■ 1080	5.23112


Codec: VP9
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.11296
■ 480	0.11713
■ 720	0.129246667
■ 1080	0.165473333

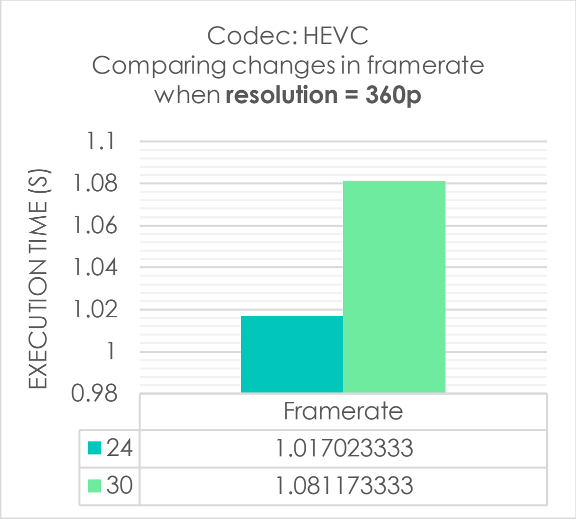

Codec: MPEG4
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.328053333
■ 480	0.375853333
■ 720	0.364003333
■ 1080	0.784393333

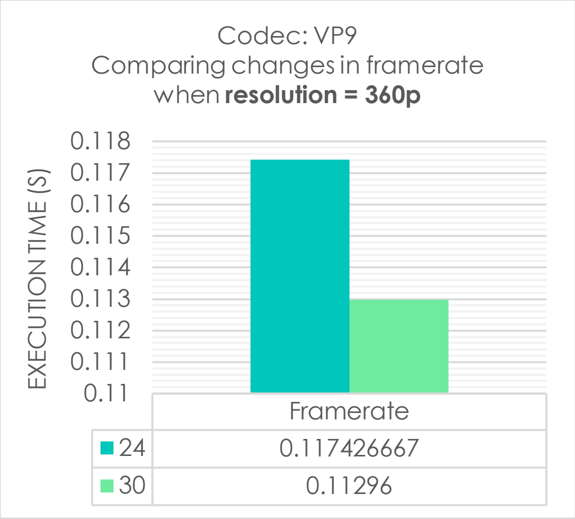

Codec: H.264
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.032633333
■ 480	0.033086667
■ 720	0.03292
■ 1080	0.032933333

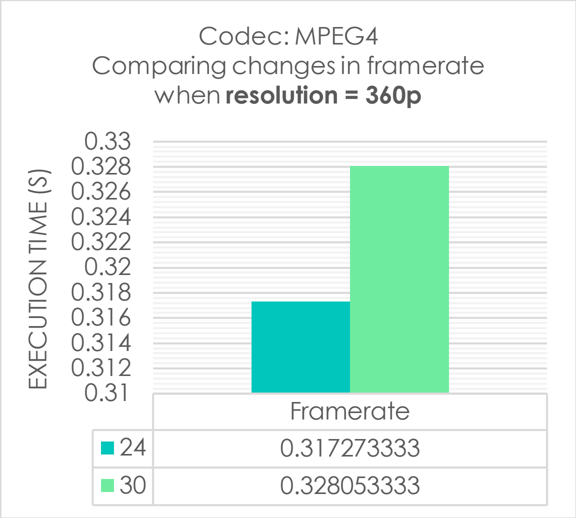
AWS-GPA: Resolution as the constant



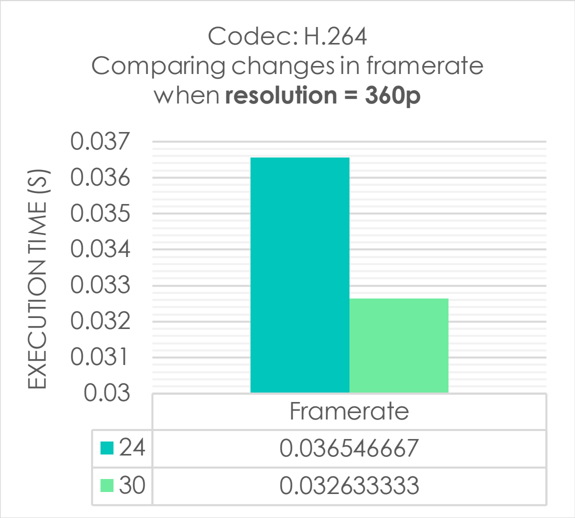
	Framerate
■ 24	1.017023333
■ 30	1.081173333



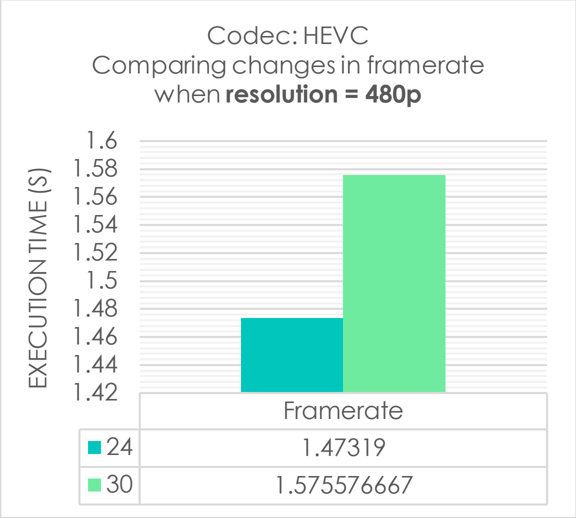
	Framerate
■ 24	0.117426667
■ 30	0.11296



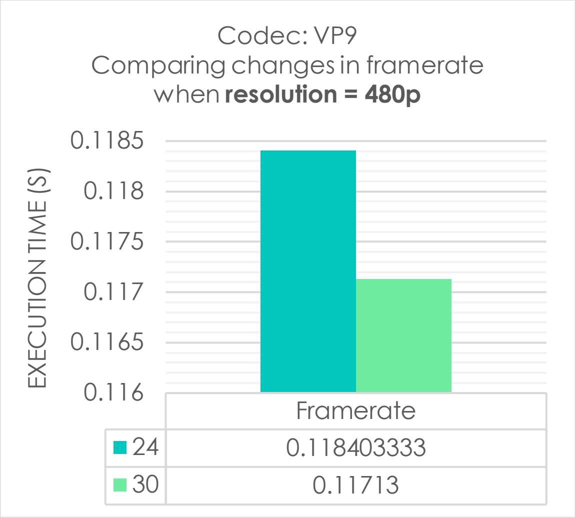
	Framerate
■ 24	0.317273333
■ 30	0.328053333



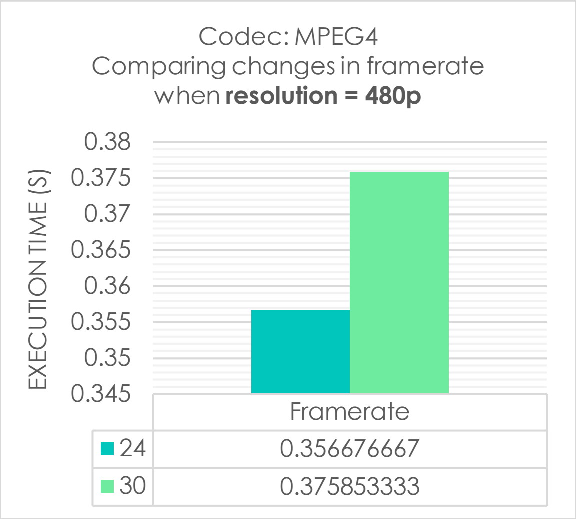
	Framerate
■ 24	0.036546667
■ 30	0.032633333



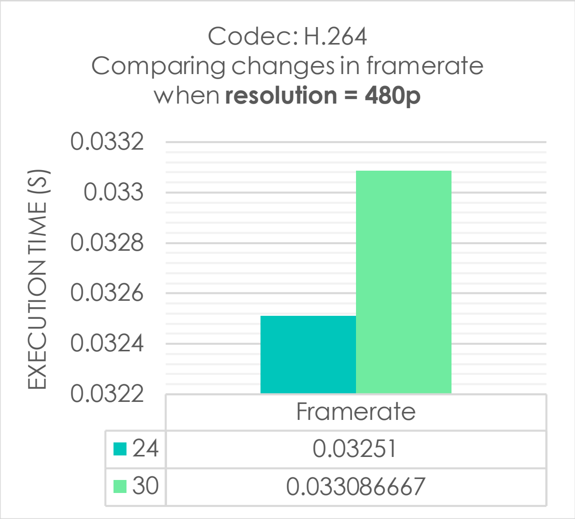
	Framerate
■ 24	1.47319
■ 30	1.575576667



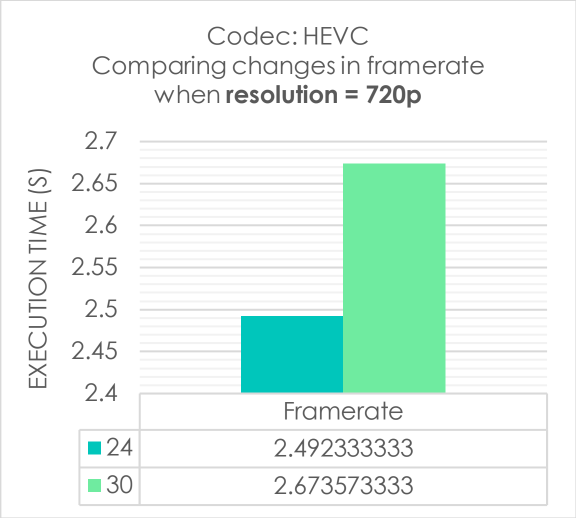
	Framerate
■ 24	0.118403333
■ 30	0.11713

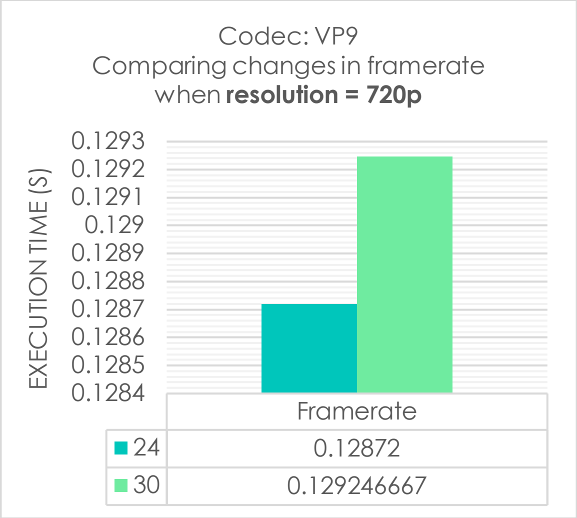


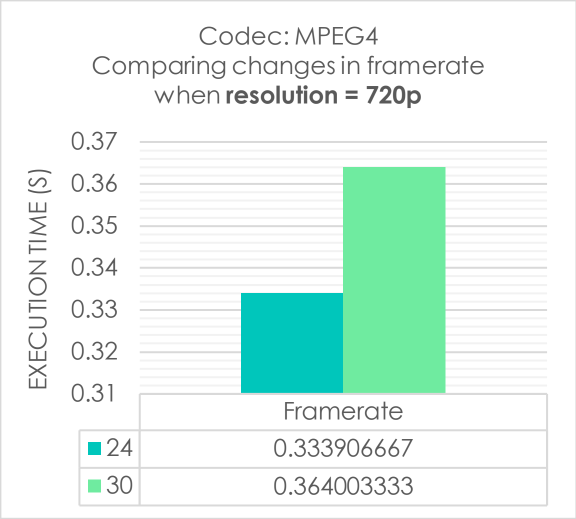
	Framerate
■ 24	0.356676667
■ 30	0.375853333

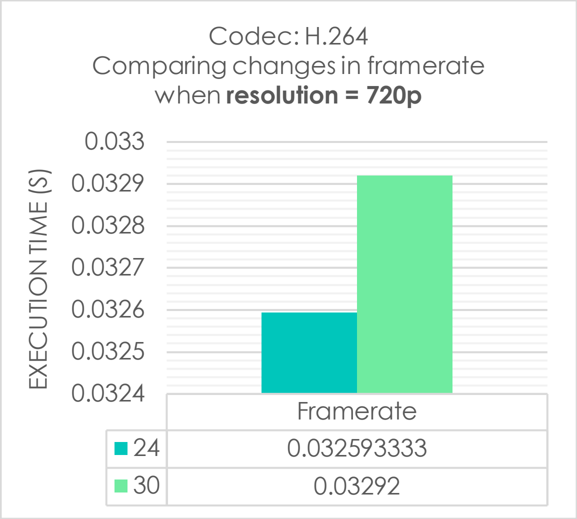


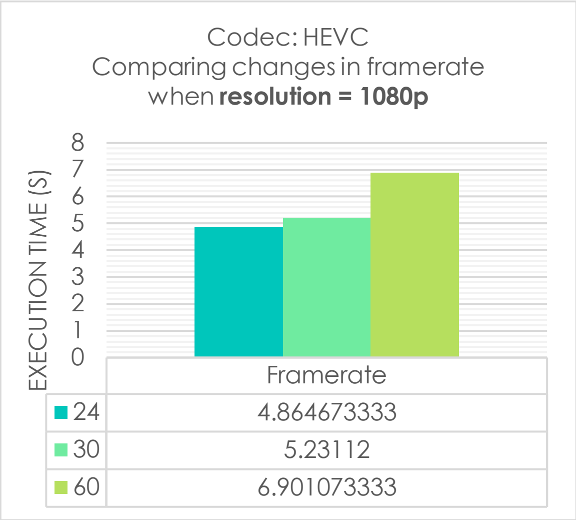
	Framerate
■ 24	0.03251
■ 30	0.033086667

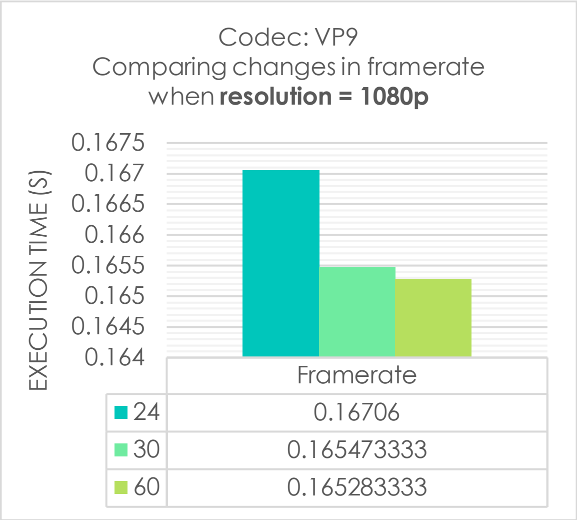

Codec: HEVC
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
2.7
2.65
2.6
2.55
2.5
2.45
2.4
Framerate
24 2.492333333
30 2.673573333

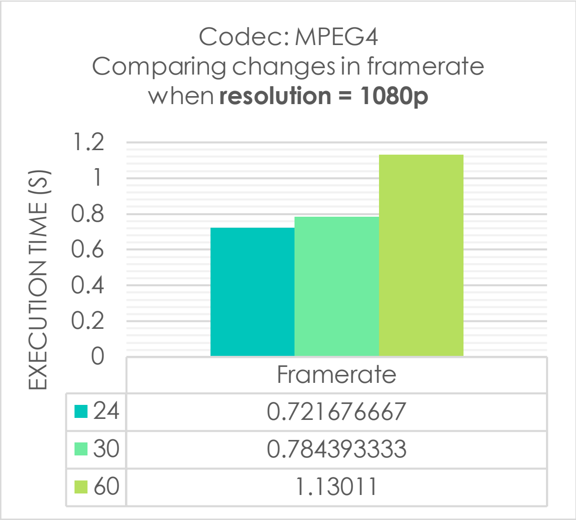

Codec: VP9
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.1293
0.1292
0.1291
0.129
0.1289
0.1288
0.1287
0.1286
0.1285
0.1284
Framerate
24 0.12872
30 0.129246667

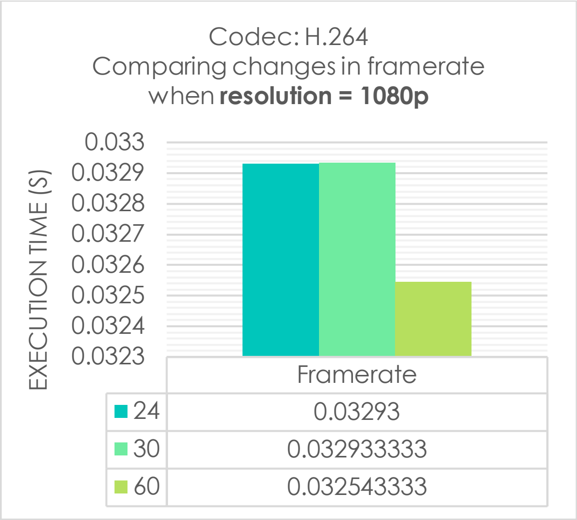

Codec: MPEG4
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.37
0.36
0.35
0.34
0.33
0.32
0.31
Framerate
24 0.333906667
30 0.364003333

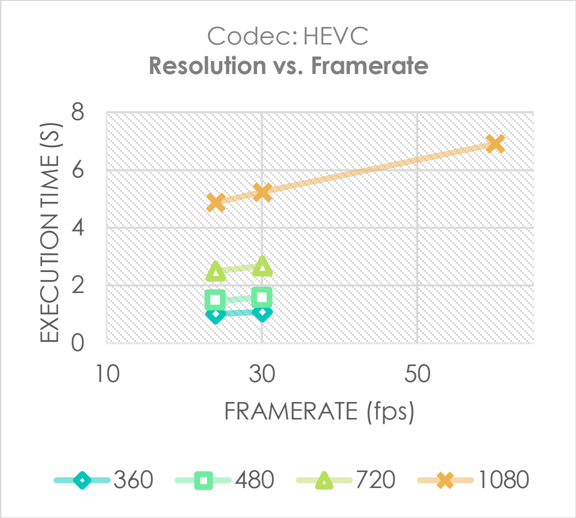

Codec: H.264
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.033
0.0329
0.0328
0.0327
0.0326
0.0325
0.0324
Framerate
24 0.032593333
30 0.03292

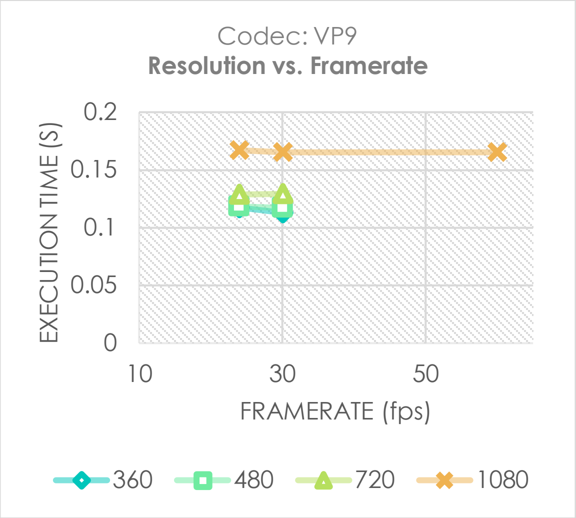

Codec: HEVC
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
8
7
6
5
4
3
2
1
0
Framerate
24 4.864673333
30 5.23112
60 6.901073333

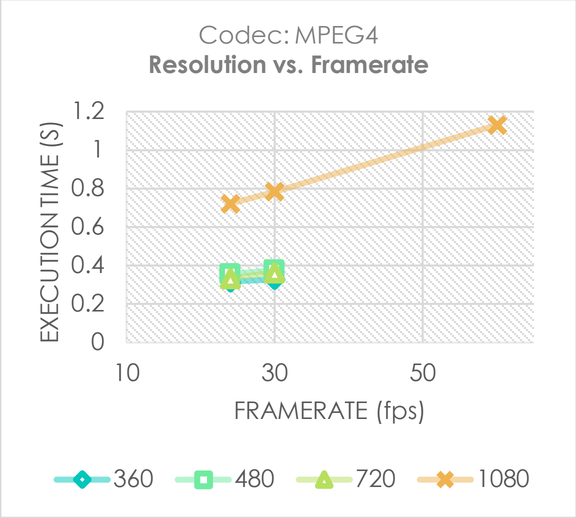

Codec: VP9
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.1675
0.167
0.1665
0.166
0.1655
0.165
0.1645
0.164
Framerate
24 0.16706
30 0.165473333
60 0.165283333

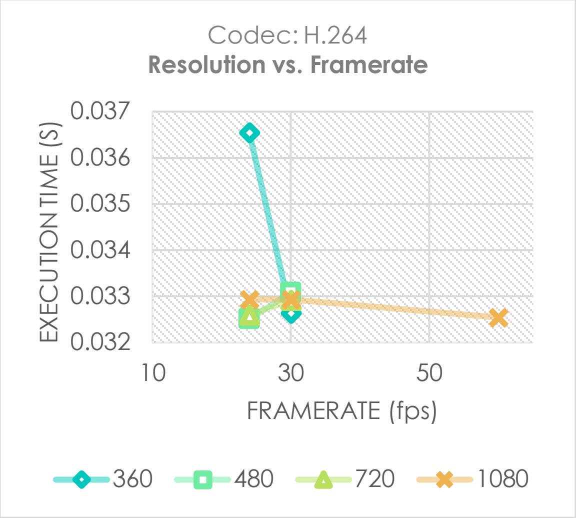

Codec: MPEG4
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
1.2
1
0.8
0.6
0.4
0.2
0
Framerate
24 0.721676667
30 0.784393333
60 1.13011

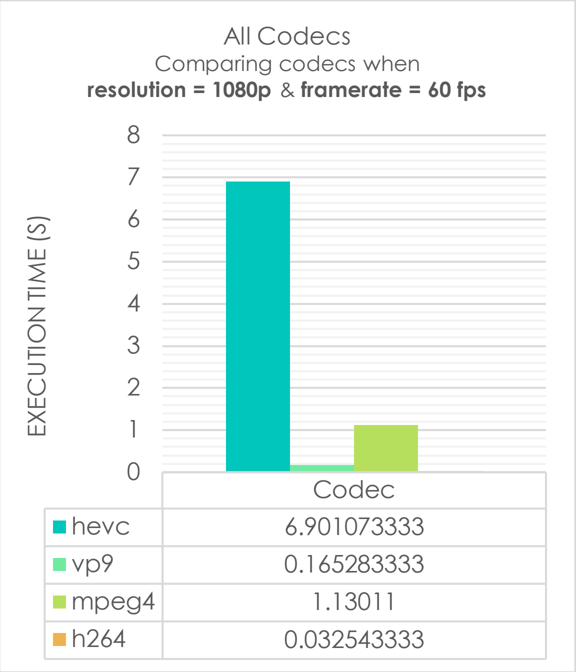

Codec: H.264
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.033
0.0329
0.0328
0.0327
0.0326
0.0325
0.0324
0.0323
Framerate
24 0.03293
30 0.032933333
60 0.032543333



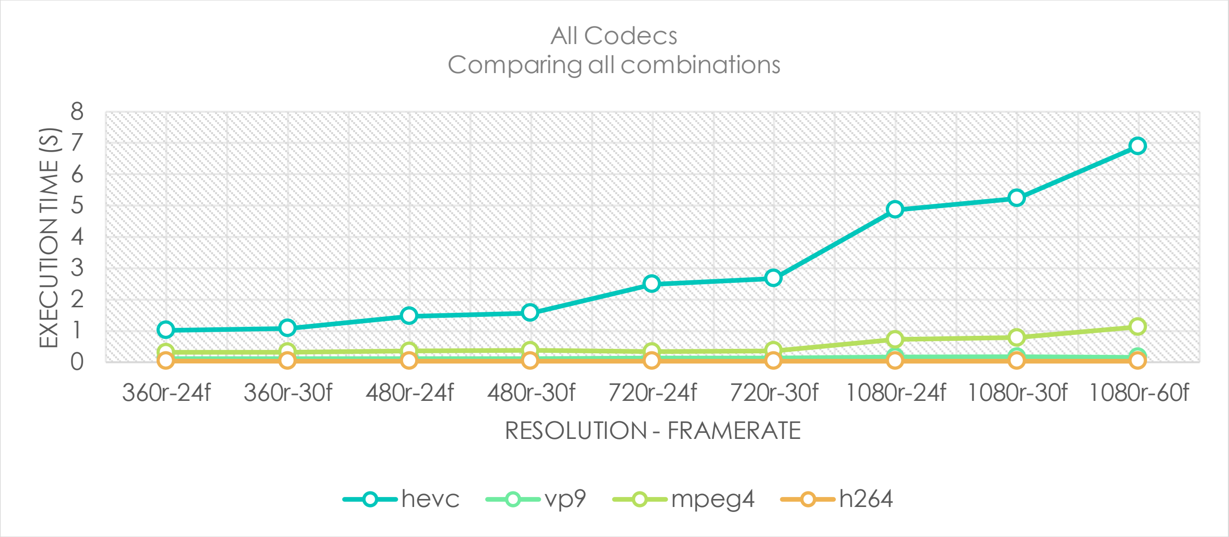




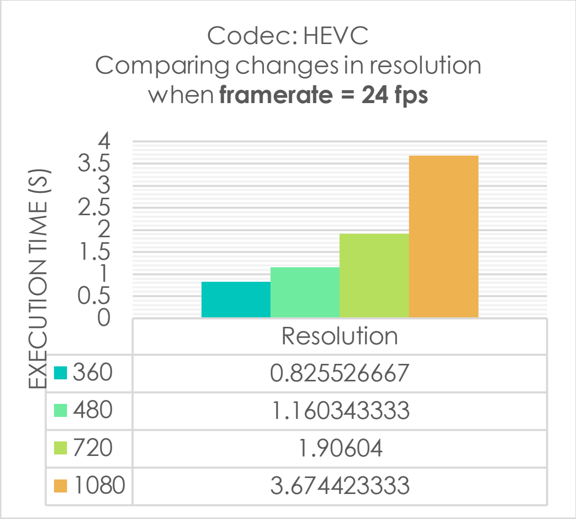




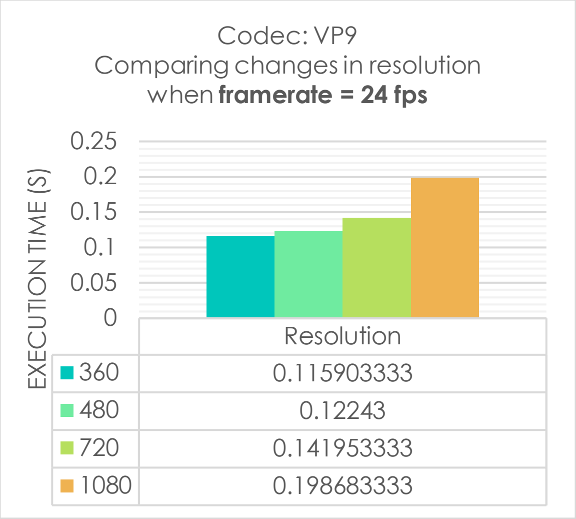
	Codec
hevc	6.901073333
vp9	0.165283333
mpeg4	1.13011
h264	0.032543333



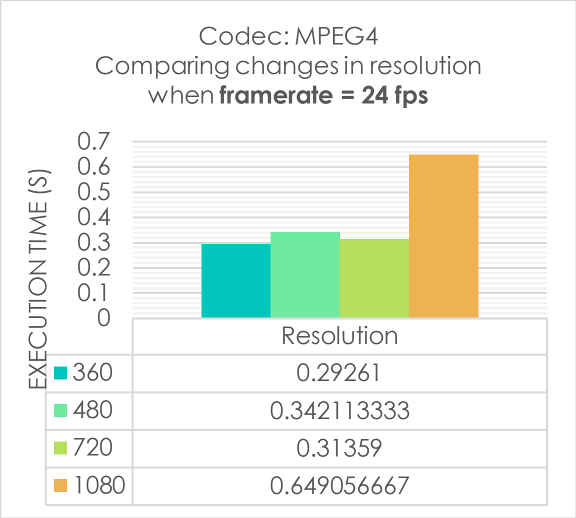
AWS-GPI: Framerate as the constant



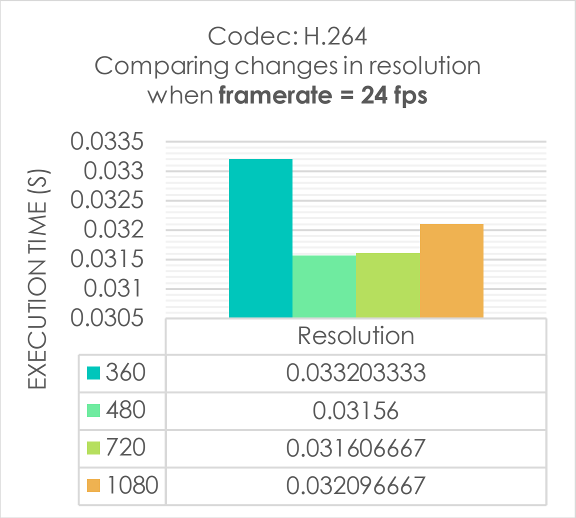
	Resolution
360	0.825526667
480	1.160343333
720	1.90604
1080	3.674423333



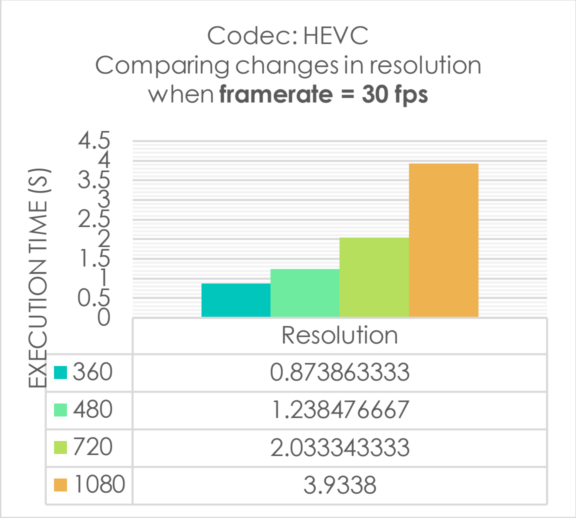
	Resolution
360	0.115903333
480	0.12243
720	0.141953333
1080	0.198683333



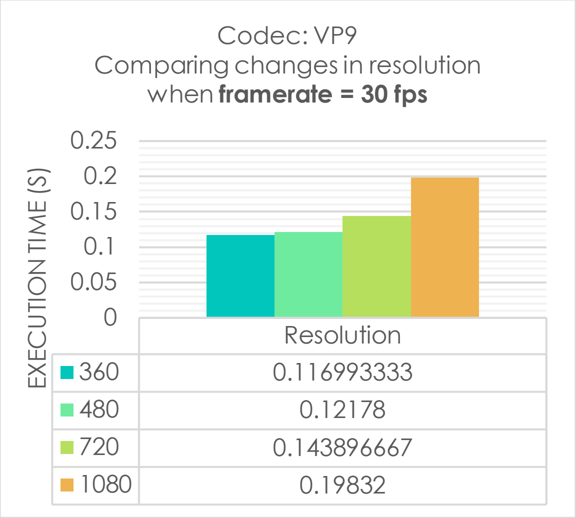
	Resolution
360	0.29261
480	0.342113333
720	0.31359
1080	0.649056667



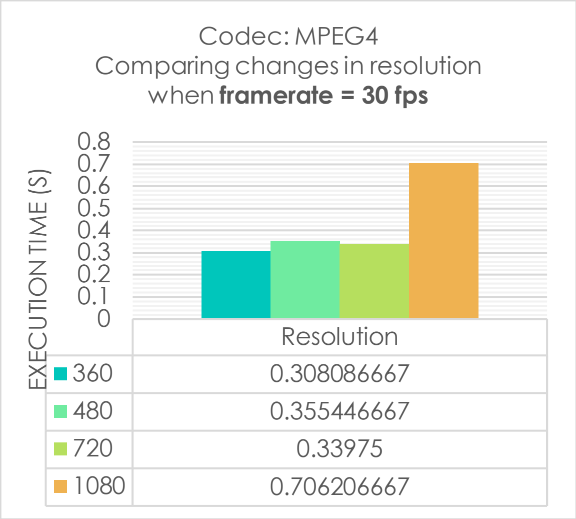
	Resolution
360	0.033203333
480	0.03156
720	0.031606667
1080	0.032096667



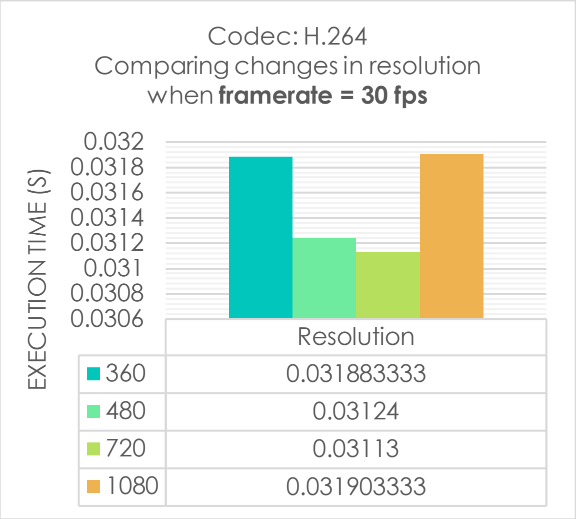
	Resolution
360	0.873863333
480	1.238476667
720	2.033343333
1080	3.9338



	Resolution
360	0.116993333
480	0.12178
720	0.143896667
1080	0.19832

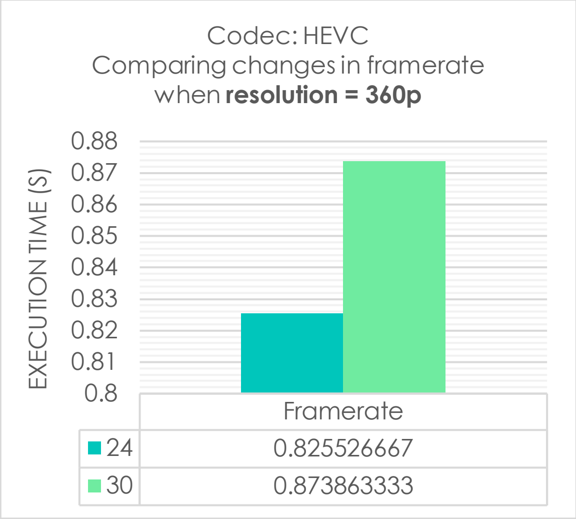


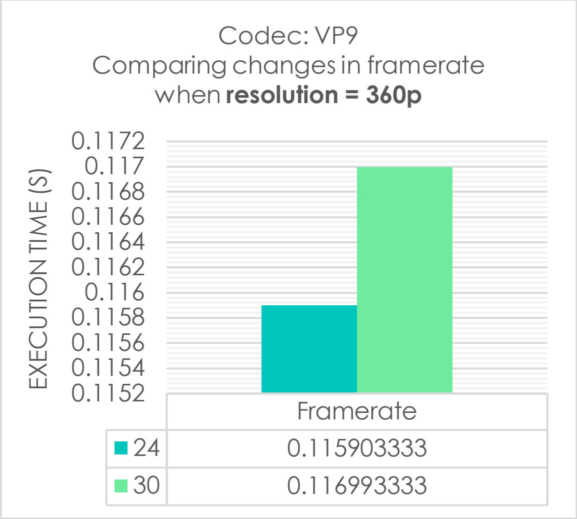
	Resolution
360	0.308086667
480	0.355446667
720	0.33975
1080	0.706206667

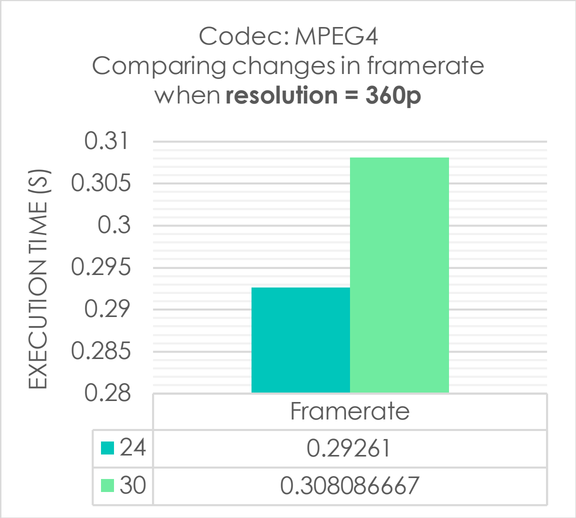


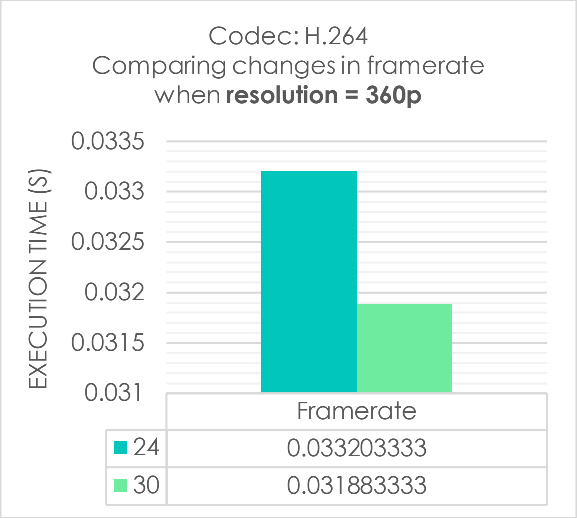
	Resolution
360	0.031883333
480	0.03124
720	0.03113
1080	0.031903333

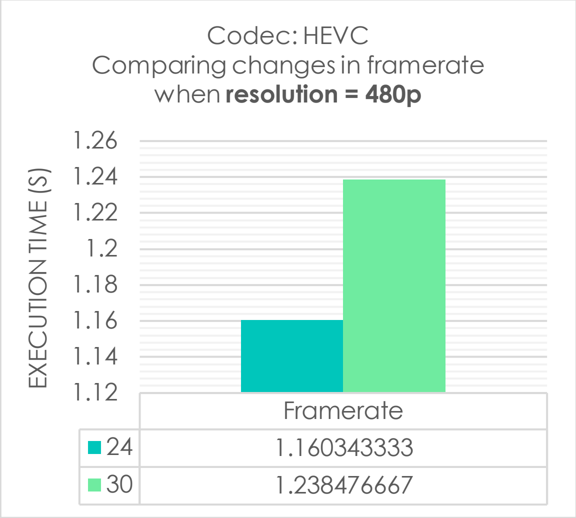
AWS-GPI: Resolution as the constant

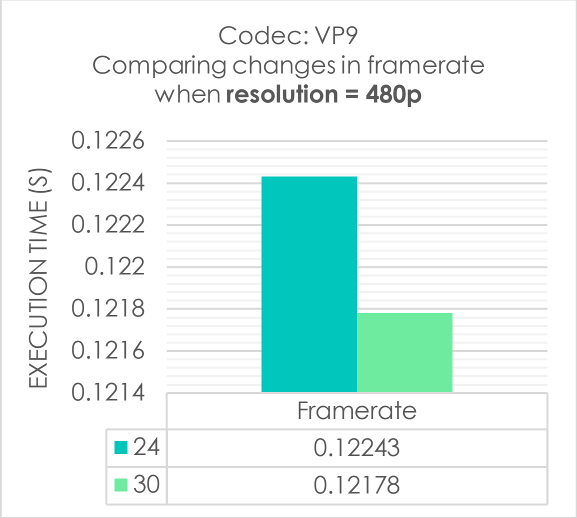


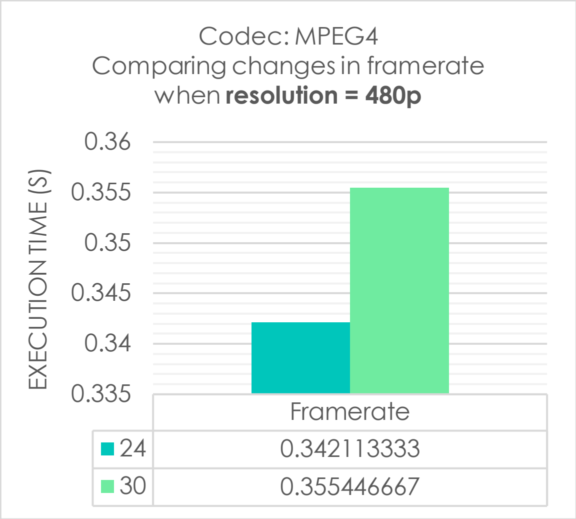


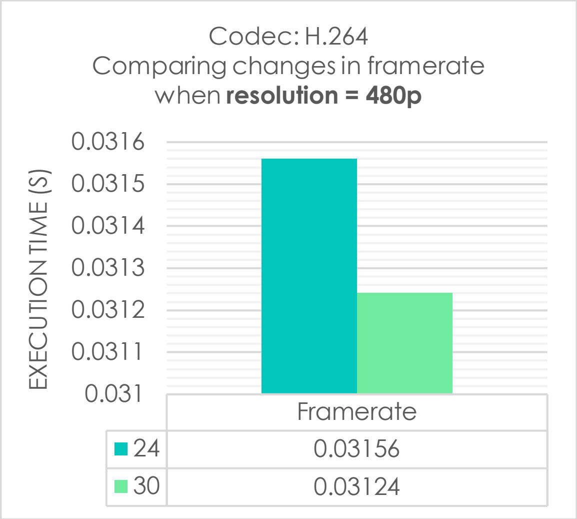


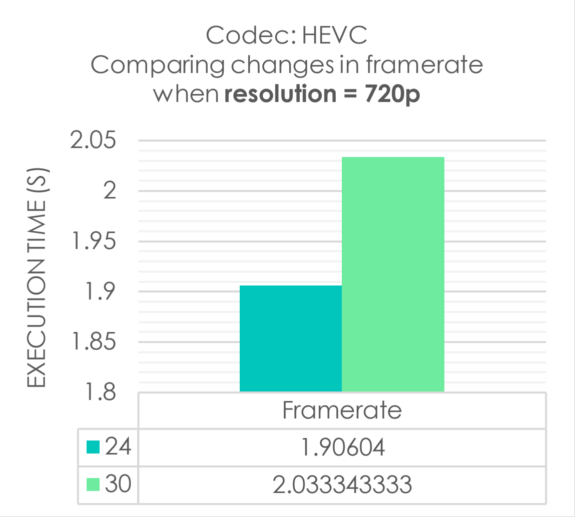


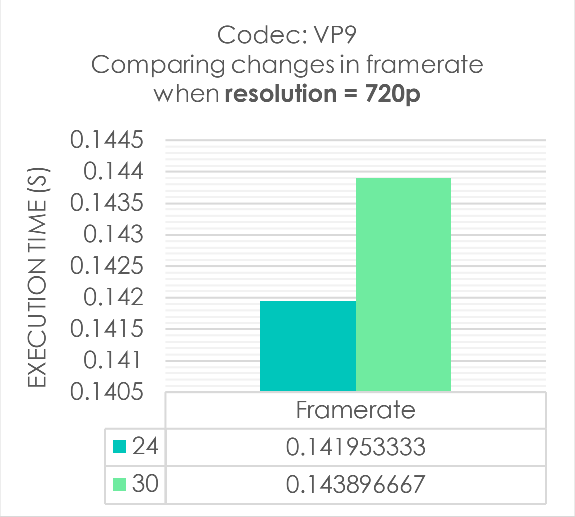


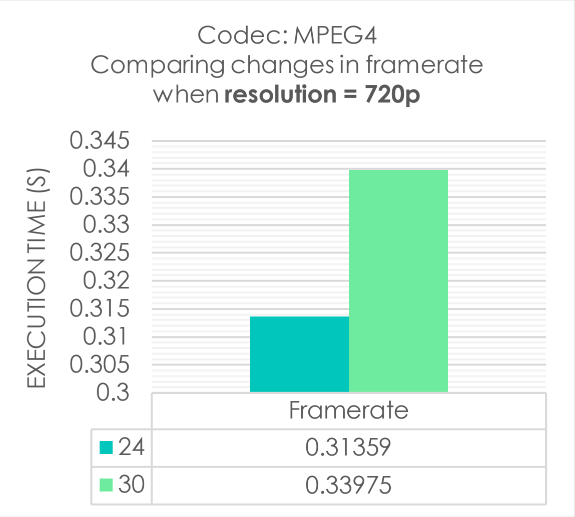


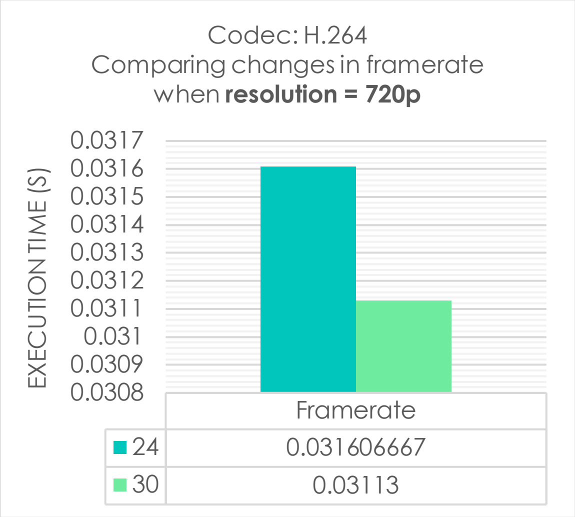


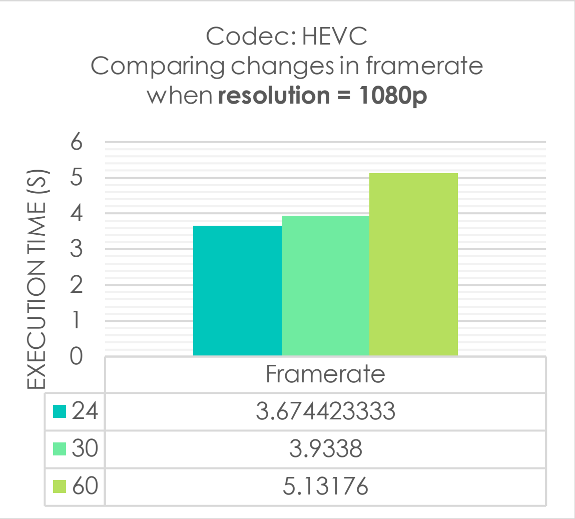


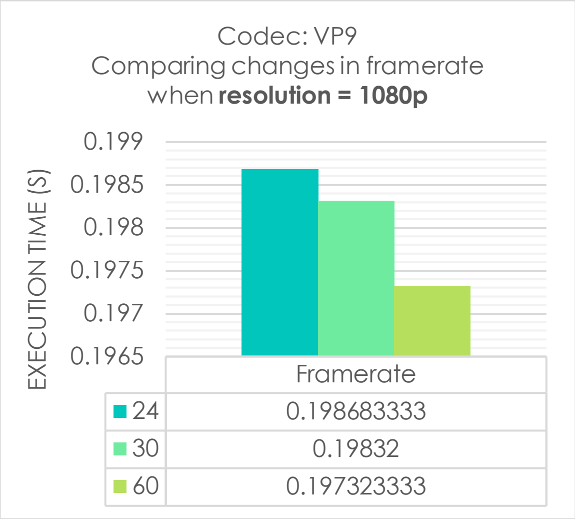

Codec: HEVC
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
2.05
2
1.95
1.9
1.85
1.8
Framerate
■ 24 1.90604
■ 30 2.033343333

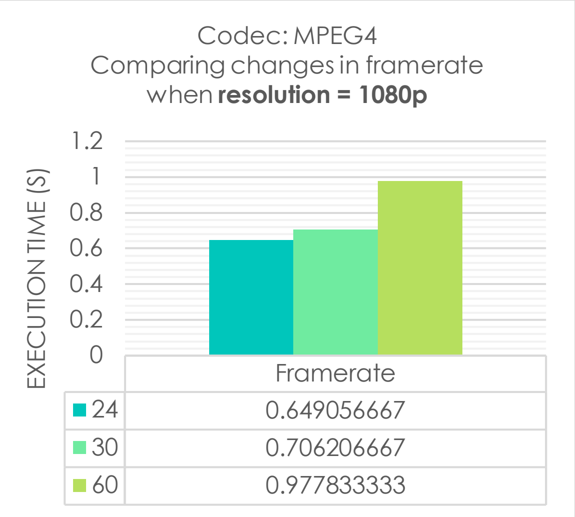

Codec: VP9
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.1445
0.144
0.1435
0.143
0.1425
0.142
0.1415
0.141
0.1405
Framerate
■ 24 0.141953333
■ 30 0.143896667

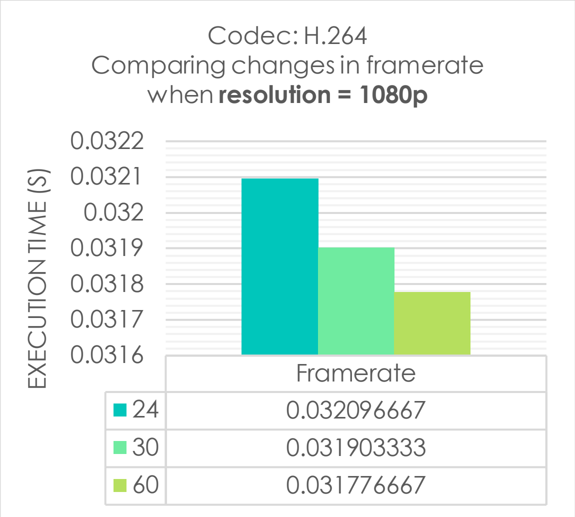

Codec: MPEG4
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.345
0.34
0.335
0.33
0.325
0.32
0.315
0.31
0.305
0.3
Framerate
■ 24 0.31359
■ 30 0.33975

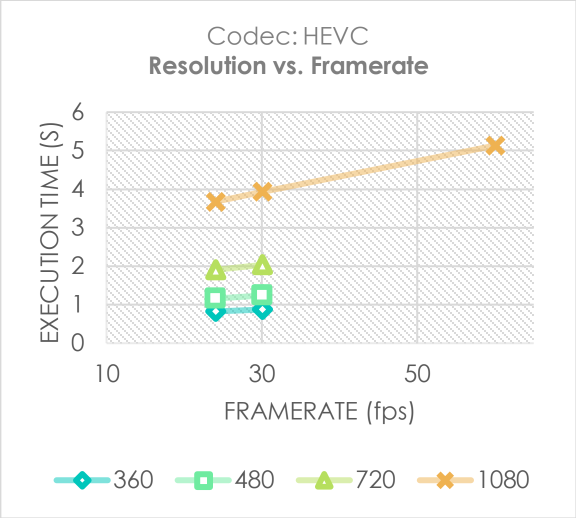

Codec: H.264
Comparing changes in framerate
when **resolution = 720p**
EXECUTION TIME (S)
0.0317
0.0316
0.0315
0.0314
0.0313
0.0312
0.0311
0.031
0.0309
0.0308
Framerate
■ 24 0.031606667
■ 30 0.03113

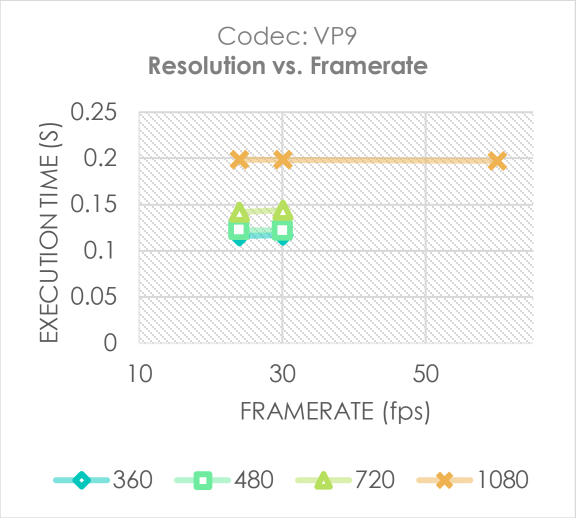

Codec: HEVC
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
6
5
4
3
2
1
0
Framerate
■ 24 3.674423333
■ 30 3.9338
■ 60 5.13176

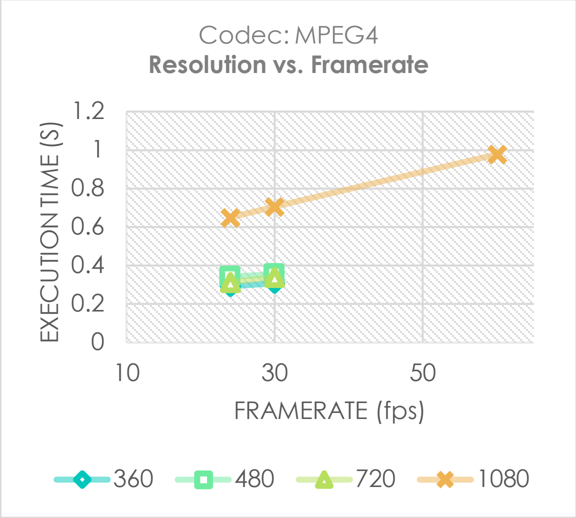

Codec: VP9
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.199
0.1985
0.198
0.1975
0.197
0.1965
Framerate
■ 24 0.198683333
■ 30 0.19832
■ 60 0.197323333

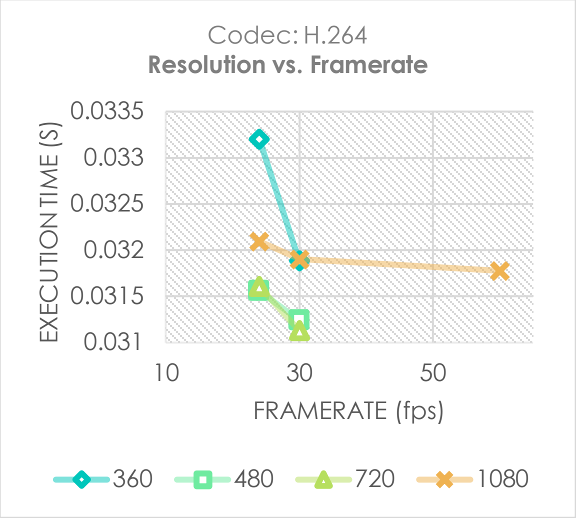

Codec: MPEG4
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
1.2
1
0.8
0.6
0.4
0.2
0
Framerate
■ 24 0.649056667
■ 30 0.706206667
■ 60 0.977833333

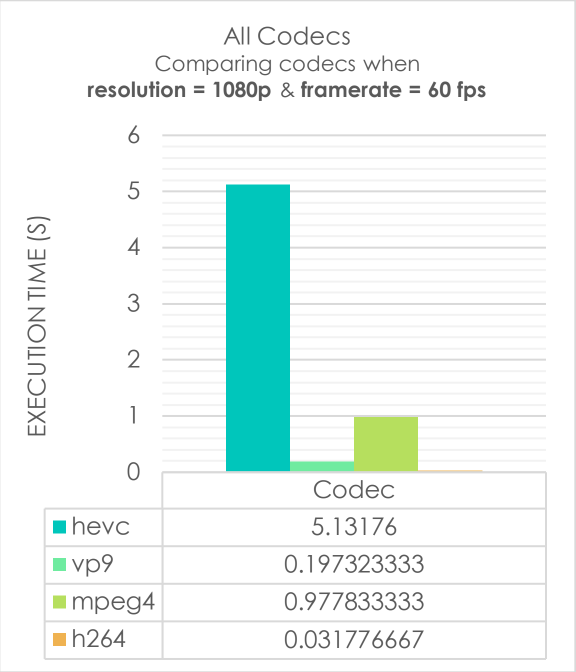

Codec: H.264
Comparing changes in framerate
when **resolution = 1080p**
EXECUTION TIME (S)
0.0322
0.0321
0.032
0.0319
0.0318
0.0317
0.0316
Framerate
■ 24 0.032096667
■ 30 0.031903333
■ 60 0.031776667



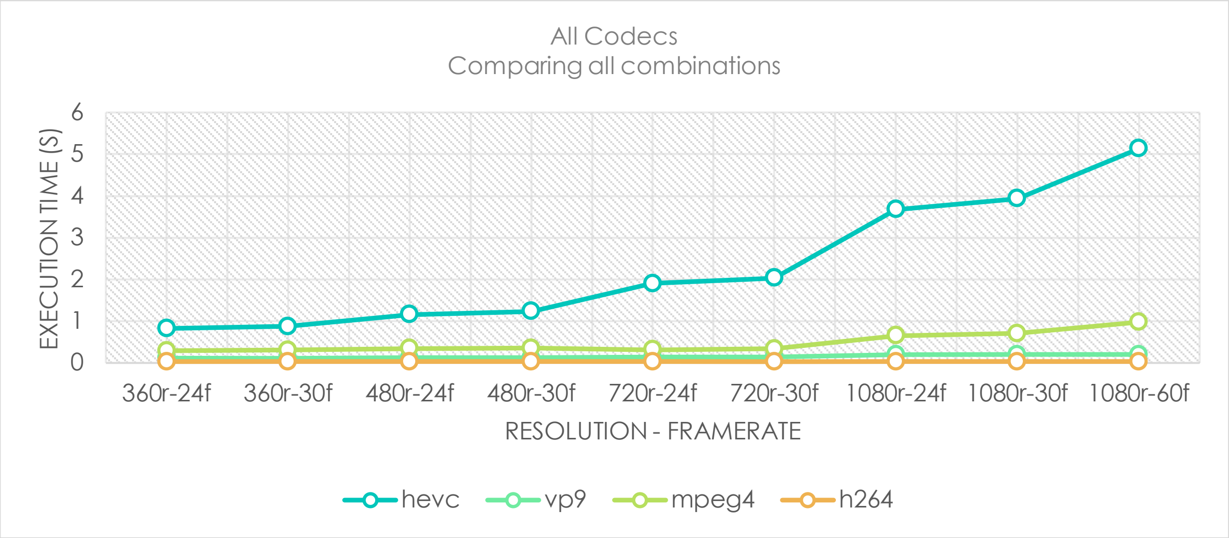




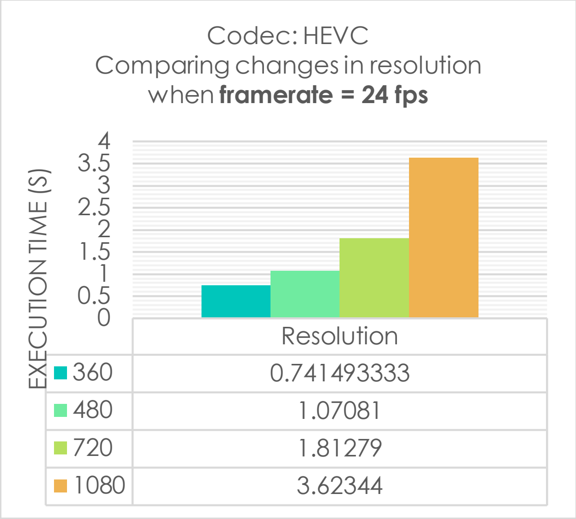




	Codec
hevc	5.13176
vp9	0.197323333
mpeg4	0.977833333
h264	0.031776667

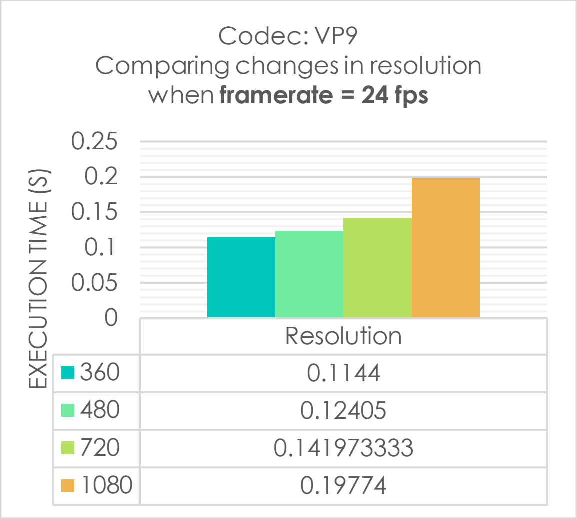


AWS-MO: Framerate as the constant



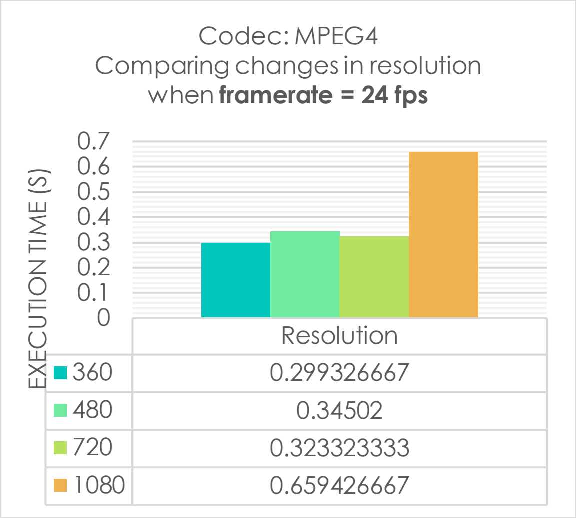
Codec: HEVC
Comparing changes in resolution
when **framerate = 24 fps**

	Resolution
■ 360	0.741493333
■ 480	1.07081
■ 720	1.81279
■ 1080	3.62344



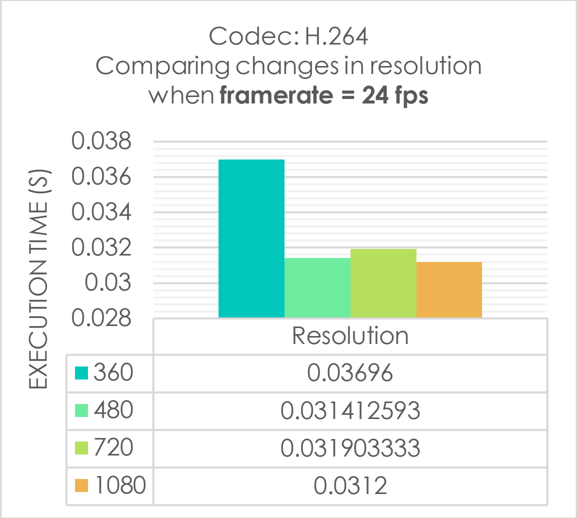
Codec: VP9
Comparing changes in resolution
when **framerate = 24 fps**

	Resolution
■ 360	0.1144
■ 480	0.12405
■ 720	0.141973333
■ 1080	0.19774



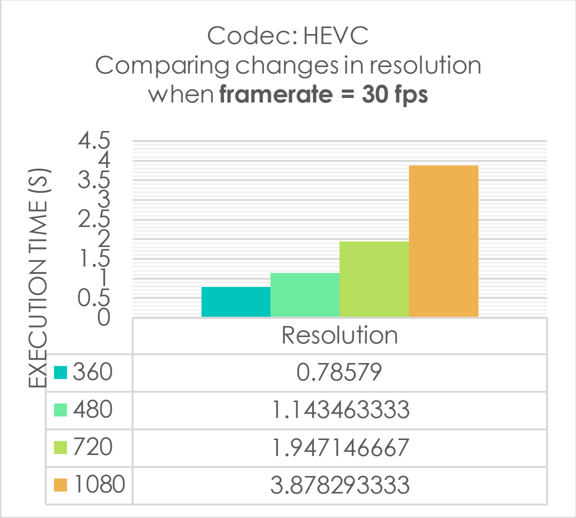
Codec: MPEG4
Comparing changes in resolution
when **framerate = 24 fps**

	Resolution
■ 360	0.299326667
■ 480	0.34502
■ 720	0.323323333
■ 1080	0.659426667



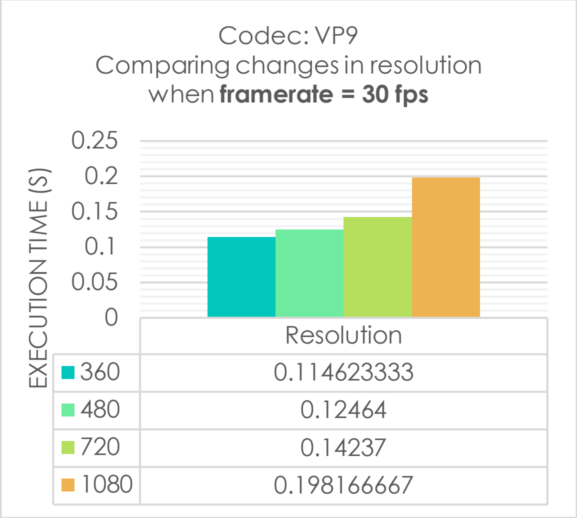
Codec: H.264
Comparing changes in resolution
when **framerate = 24 fps**

	Resolution
■ 360	0.03696
■ 480	0.031412593
■ 720	0.031903333
■ 1080	0.0312



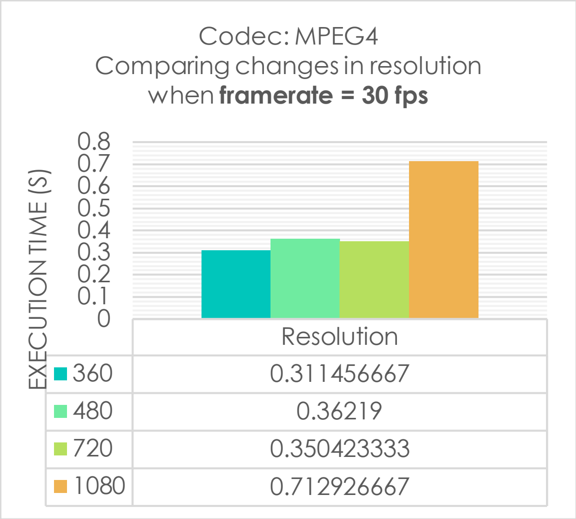
Codec: HEVC
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.78579
■ 480	1.143463333
■ 720	1.947146667
■ 1080	3.878293333



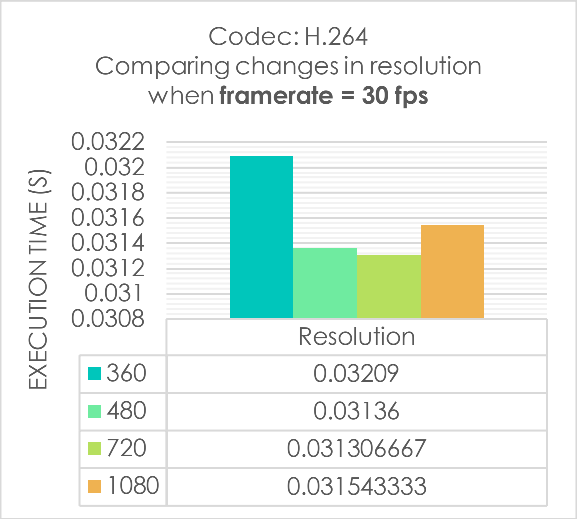
Codec: VP9
Comparing changes in resolution
when **framerate = 30 fps**

	Resolution
■ 360	0.114623333
■ 480	0.12464
■ 720	0.14237
■ 1080	0.198166667



Codec: MPEG4
Comparing changes in resolution
when **framerate = 30 fps**

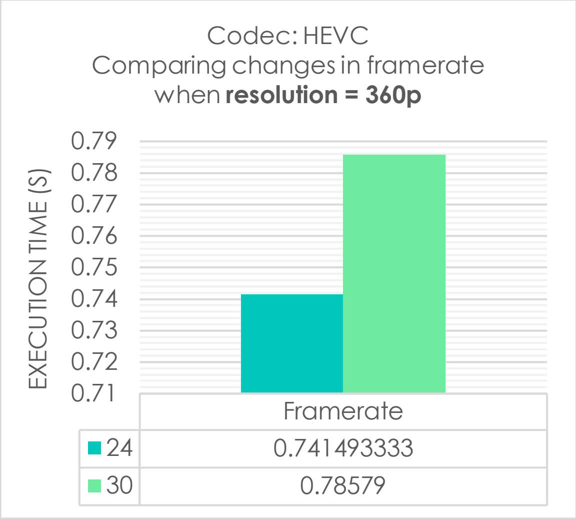
	Resolution
■ 360	0.311456667
■ 480	0.36219
■ 720	0.350423333
■ 1080	0.712926667

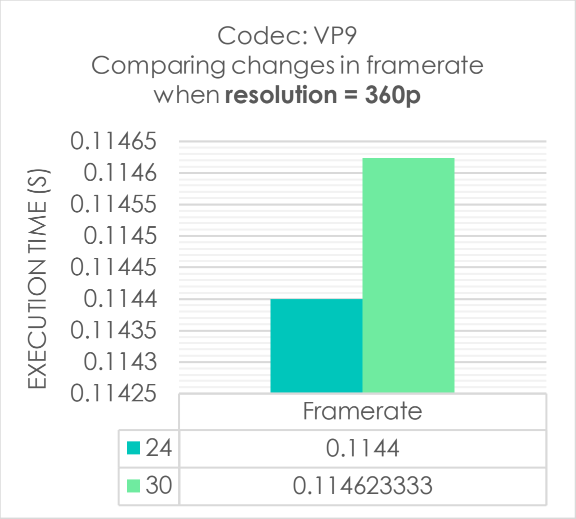


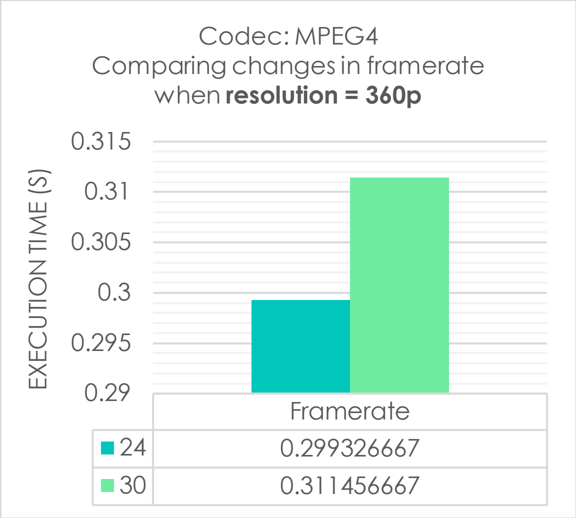
Codec: H.264
Comparing changes in resolution
when **framerate = 30 fps**

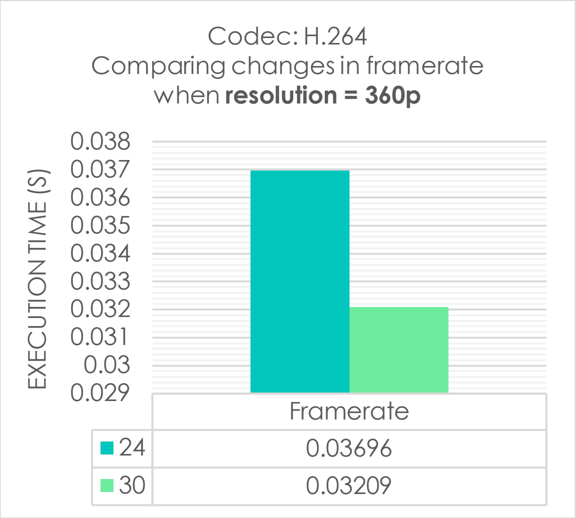
	Resolution
■ 360	0.03209
■ 480	0.03136
■ 720	0.031306667
■ 1080	0.031543333

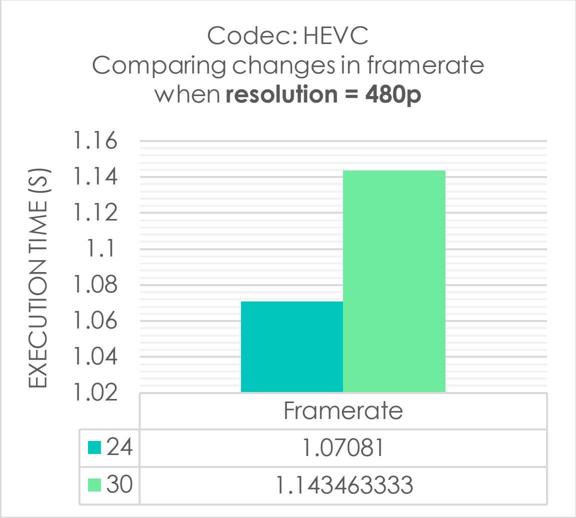
AWS-MO: Resolution as the constant

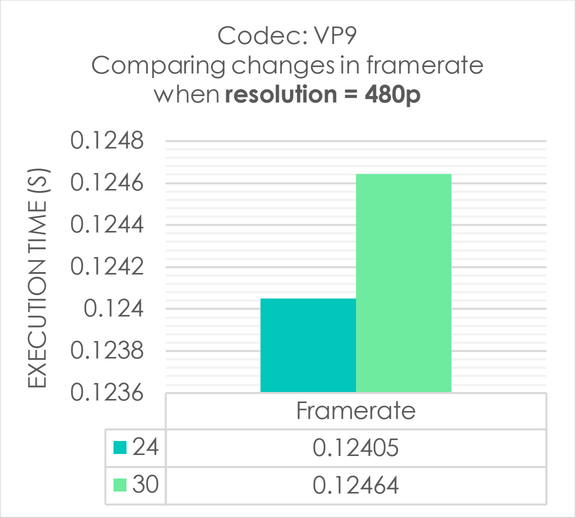


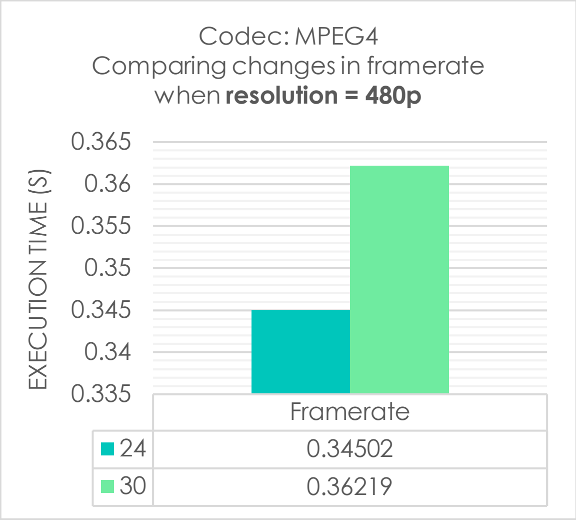


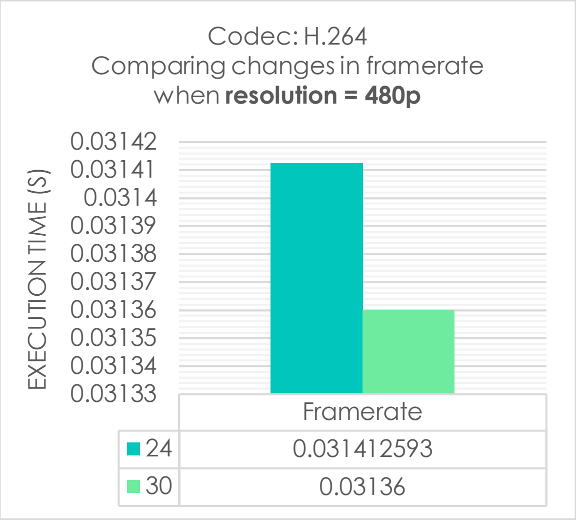


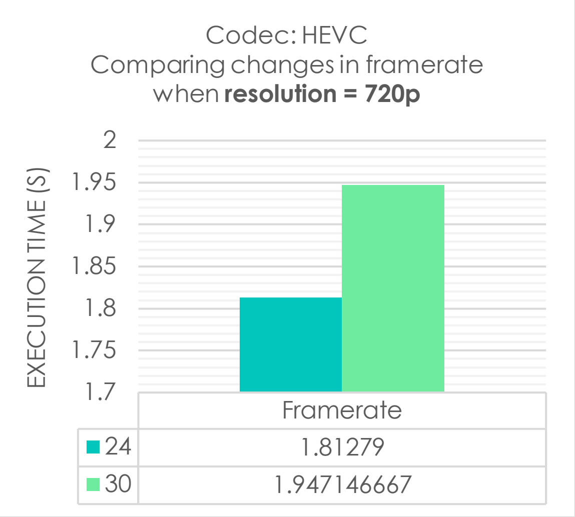






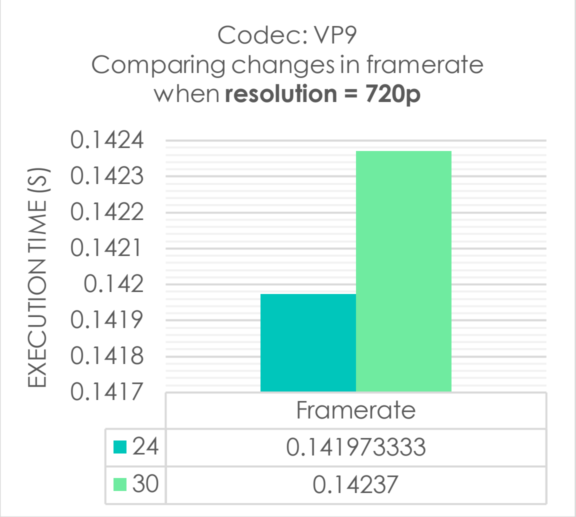






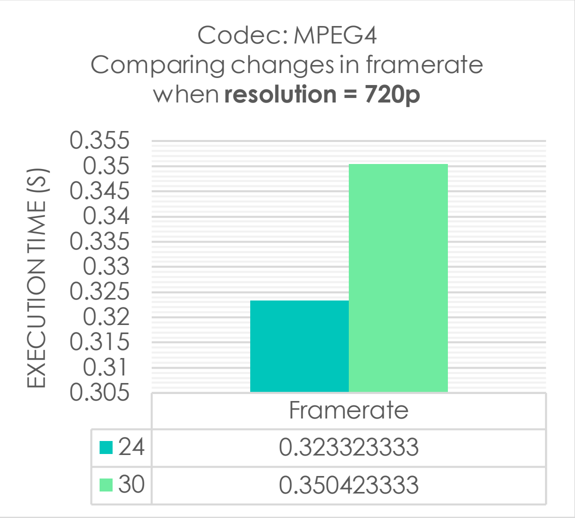
Codec: HEVC
Comparing changes in framerate
when **resolution = 720p**

	Framerate
■ 24	1.81279
■ 30	1.947146667



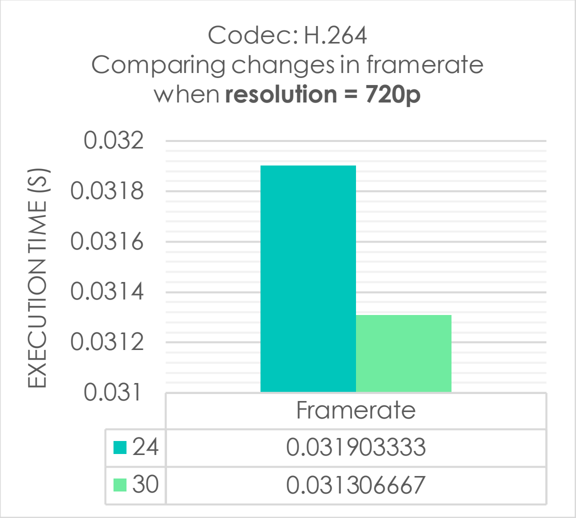
Codec: VP9
Comparing changes in framerate
when **resolution = 720p**

	Framerate
■ 24	0.141973333
■ 30	0.14237



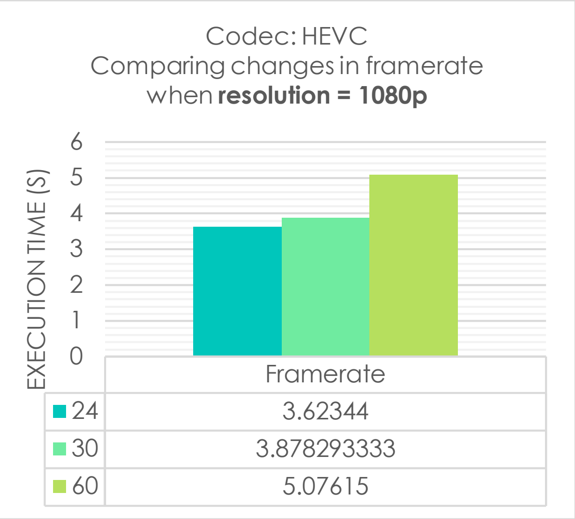
Codec: MPEG4
Comparing changes in framerate
when **resolution = 720p**

	Framerate
■ 24	0.323323333
■ 30	0.350423333



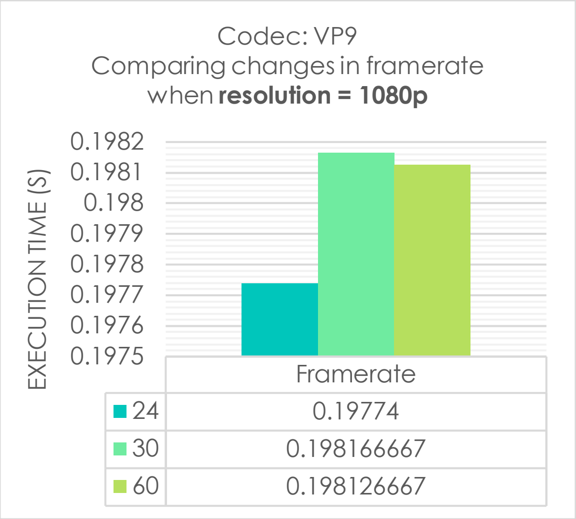
Codec: H.264
Comparing changes in framerate
when **resolution = 720p**

	Framerate
■ 24	0.031903333
■ 30	0.031306667



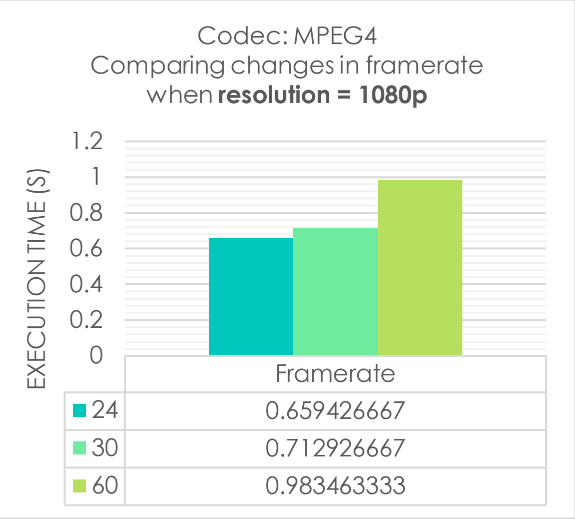
Codec: HEVC
Comparing changes in framerate
when **resolution = 1080p**

	Framerate
■ 24	3.62344
■ 30	3.878293333
■ 60	5.07615



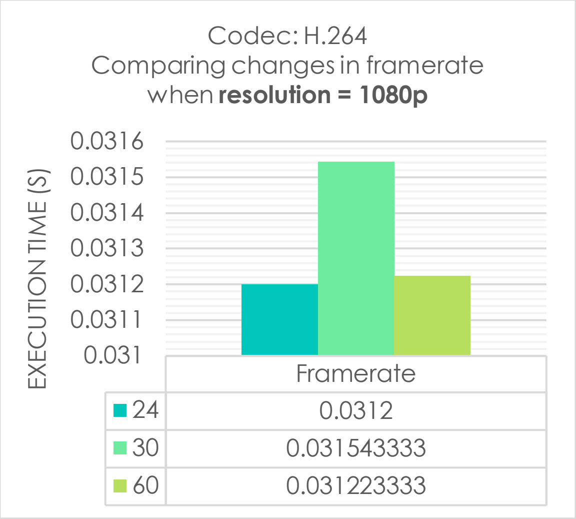
Codec: VP9
Comparing changes in framerate
when **resolution = 1080p**

	Framerate
■ 24	0.19774
■ 30	0.198166667
■ 60	0.198126667



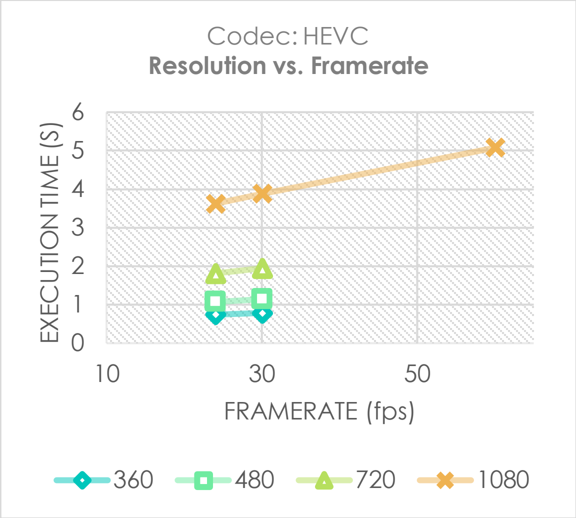
Codec: MPEG4
Comparing changes in framerate
when **resolution = 1080p**

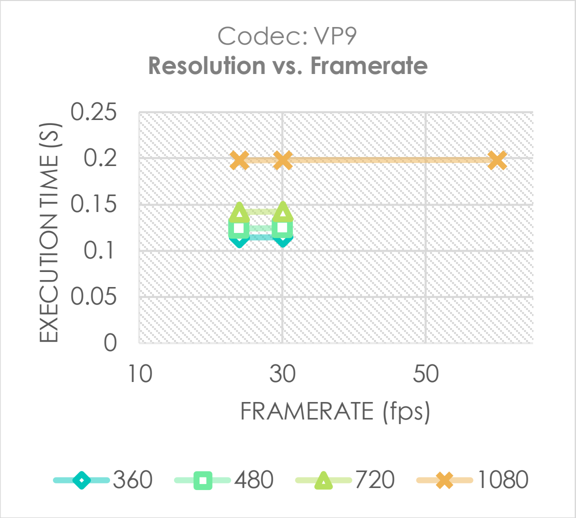
	Framerate
■ 24	0.659426667
■ 30	0.712926667
■ 60	0.983463333

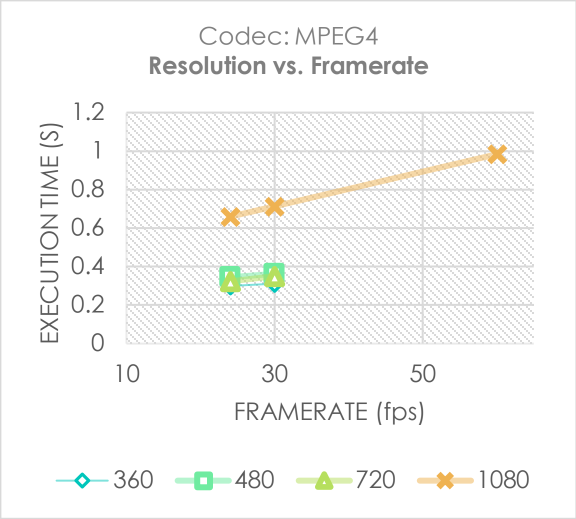


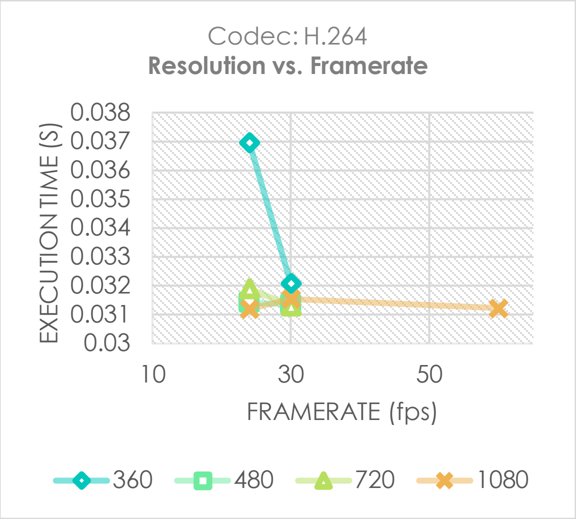
Codec: H.264
Comparing changes in framerate
when **resolution = 1080p**

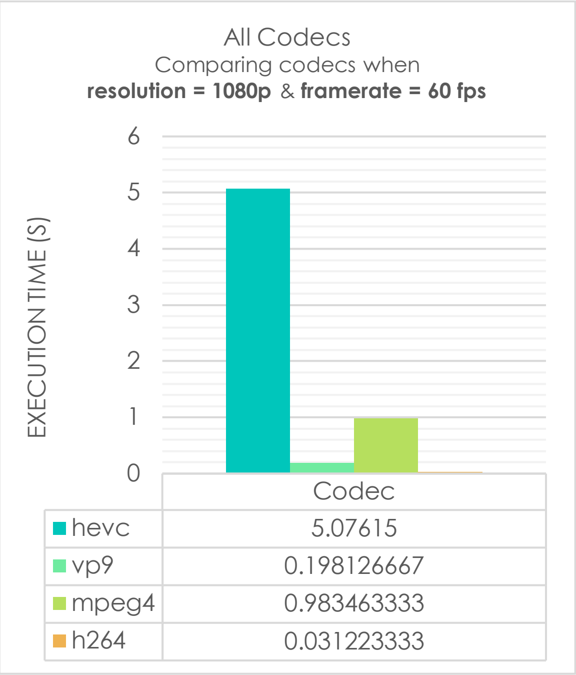
	Framerate
■ 24	0.0312
■ 30	0.031543333
■ 60	0.031223333



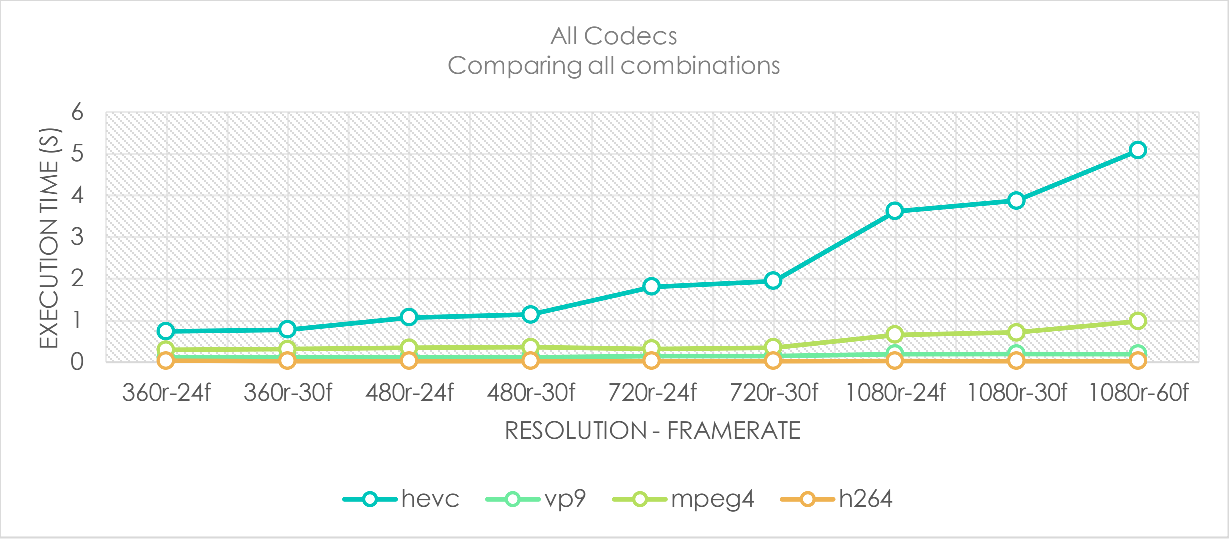








	Codec
■ hevc	5.07615
■ vp9	0.198126667
■ mpeg4	0.983463333
■ h264	0.031223333



D. Evaluation

- **CC**
  - In all codec scripts, the Resolution vs. Framerate graphs appear very similar. It seems that there is not much difference between the results for 24 fps and 30 fps in general for the 360p, 480p, and 720p resolutions. On the other hand, for the HEVC and MPEG4, there is a linear relationship for the 1080p resolution as the execution time increases when the framerate increases. There are similarities until the H.264 graphs, which has a lot of variance in results.
  - In particular for the H.264 Resolution vs. Framerate graphs, it seems the 360p resolution's changes from 24 fps to 30 fps is a downward slope. This may be caused by "down sampling" the resolution or framerates. In contrast, the changes in framerate for the 1080p resolution could be "up sampling" as it tries to transcode to a higher quality. Furthermore, the conversion trends for the 720p resolution and 30 fps may be because the standardized format of the videos in the dataset is 720p, 30 fps, and H.264 codec. Therefore, the script does not have the overhead of converting to a different resolution, framerate, or codec for some combinations.
  - Since CC-Large's HEVC run is incomplete, I will only compare the results from the VP9, MPEG4, and H.264 scripts.
  - <u>VP9</u>
    - For this codec, the fastest machine type was CC-Medium with 0.1907 seconds. The next fastest was CC-Large with 0.1924 seconds. This seems odd considering CC-Medium and CC-Large have less CPUs than CC-XLarge, which had an execution time of 0.2177 seconds. This occurrence also appears in the lowest conversion combination, which is 360p and 24 fps. CC-Medium ran 0.1666 seconds whereas CC-XLarge ran 0.1896 seconds. These results may be caused by Chameleon's resource allocations. Overall, CC-Medium had the fastest execution times, and CC-2xLarge had the slowest execution times.
  - <u>MPEG4</u>
    - For this codec, the fastest machine type was CC-2xLarge with 0.4212 seconds. The CC-XLarge and CC-Large follow closely with 0.4440 and 0.4459 seconds respectively. In general, CC-2xLarge and CC-XLarge have similar execution times overall for this codec.
  - <u>H.264</u>
    - For this codec, the fastest machine type was CC-Medium with 0.0969 seconds for the 720p and 30 fps conversion. The next fastest execution time was 0.0981 seconds for CC-XLarge. Again, this result seems odd in regard to the number of CPUs. Overall, it seems CC-XLarge has a consistent range of 0.097-0.099 seconds, and CC-Medium has a range of 0.095-0.105 seconds. Therefore, either CC-Medium or CC-XLarge has the fastest execution times, but CC-2xLarge has the slowest execution times.
- **AWS**
  - Similar to the CC machine types, the graphs and trends appear almost the same, with H.264 codec being the outlier again. Because this trend is consistent between CC and AWS, it may confirm that the variant results were not caused by Chameleon's resources. As previously mentioned, the downward slope for the 360p resolution as it changes from 24 fps to 30 fps may be caused by down sampling the resolution or framerates. Also, the changes in framerate for the 1080p resolution could be up sampling. In brief, since the H.264 graphs for both CC and AWS appear similar, there is evidence that the cloud resources may not be affecting the results but the FFPMEG commands itself in transcoding and up/down-sampling.
  - <u>HEVC</u>
    - For this codec, the fastest execution time was 1.759 seconds with AWS-CO. The slowest execution time was 2.673 seconds from AWS-GPA.

- VP9
    - For this codec, the fastest execution time was 0.129 seconds with AWS-GPA. The slowest execution time was 0.131 seconds with AWS-CO.
- MPEG4
    - For this codec, the fastest execution time was 0.320 seconds with AWS-CO. The slowest execution time was 0.364 seconds with AWS-GPA.
- H.264
    - For this codec, the fastest execution time was 0.028 seconds with AWS-CO. The slowest execution time was 0.032 seconds with AWS-GPA.
- Since this part has a lot of data, I put together a minimum-maximum range chart for each of the AWS machines:

Execution Time Range of Every Resolution and Framerate (Min-Max)				
	HEVC	VP9	MPEG4	H.264
CO	0.660 - 4.729	0.107 - 0.188	0.272 - 0.902	0.028 - 0.040
GPA	1.017 - 6.901	0.112 - 0.165	0.317 - 1.13	0.032 - 0.036
GPI	0.825 - 5.131	0.115 - 0.197	0.292 - 0.977	0.031 - 0.332
MO	0.741 - 5.076	0.114 - 0.198	0.299 - 0.983	0.031 - 0.036

- Based on these results, AWS-CO generally is the fastest machine type when transcoding because it has the fastest times for each codec type. On the other hand, AWS-GPA generally is the slowest machine type.
- AWS-GPA is also slower than AWS-GPI for most of the conversion combinations, which indicates that the Intel CPU is better than the AMD CPU in the general-purpose instances.

IV. Conclusion

The results from these projects may provide some insight into efficiency in the Fire Detection application and video transcoding. Based on these experiments, benchmarking using AWS provided much faster execution times than CC, which is expected due to Amazon's resources. Because of the resource disparity, I compared the virtual machines within each cloud platform.

Both CC and AWS have similar trends with their results. Many of the output were similar in execution time and demonstrated a linear trend with increasing framerate or resolution. Moreover, for the video benchmarking, the codecs with the lowest to highest execution times were H.264, VP9, MPEG4, then HEVC for each machine type. This indicates that it is cost efficient to first standardize a dataset of videos with the prevalent H.264 codec and only transcode the framerate and resolution because it eliminates some overhead in conversion. The H.264 codec may have some variant results due to up/down sampling, but it overall resulted in the fastest execution times. Otherwise, VP9 is the next best option for codec conversion if the dataset is not standardized.

Furthermore, it seems that some machine types on each cloud platform does not majorly improve performance, with only tenths of seconds in difference. Therefore, an individual can choose a cheaper machine type on AWS for example and still obtain the same efficient performance. Although transcoding to a particular conversion combination may save seconds in execution, the time saved may be critical during a fire with the Fire Detection app or during on-demand streaming for a video.

To conclude my overall experience, some problems I faced during these projects includes internet connection issues, terminating instances, and time-consuming runs. Despite these issues, I believe I have collected some notable data over the months and have gained more understanding in cost-efficiency in virtual machines and cloud resources. Overall, I have obtained more knowledge in virtual machines, machine learning, bash scripting, and AWS than my prior understanding of those topics.

V. References

I would like to thank Chavit Denninart, Ali Mokhtari, Davood Samani, Razin Hussain, and Alireza Pakravan for their help and guidance during these projects and Dr. Mohsen Amini for the opportunity to work in the HPCC Lab.

Fire App Benchmark
Excel files:
https://www.dropbox.com/sh/9xgz8bu648sazu5/AACRClG09M15nI6uhv5_Yae1a?dl=0

Video Benchmark
Excel files:
https://www.dropbox.com/sh/48bk4184t73zfop/AADUlAf8_Vc7ed16Abna2IqIa?dl=0