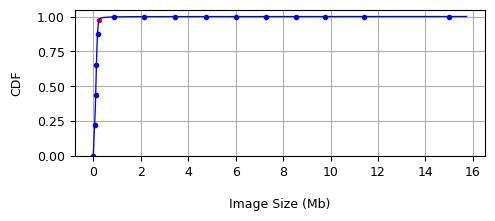
GreenFaaS: The motivation of the project

The idea behind the GreenFaaS project is to propose a **new design of the FaaS (Function as a Service) platform that reduces energy consumption during the execution of functions while maintaining performance as much as possible (execution time, result quality, and resource consumption)**. The first step of the project is to show that for a task (a function), there are alternative implementations whose execution can produce different but acceptable results, with different energy consumption and resource consumption. To do this, we started by implementing state-of-the-art benchmarks, by designing and comparing different alternatives implementations.

This report presents the analysis of the execution of [SeB’s](https://github.com/spcl/serverless-benchmarks) benchmarks. The tests were performed on the **OpenWhisk** platform, on a host machine with an **8-core processor** on a single socket, **16 GB** **of RAM**, all running under **Ubuntu 22.04.4 LTS**. We used the “**performance**” mode of the governor, setting the frequency of all cores to the maximum frequency, and used Amazon S3 as the remote storage service. The benchmarks that have already been implemented are the following:

1. **210.thumbnailer**

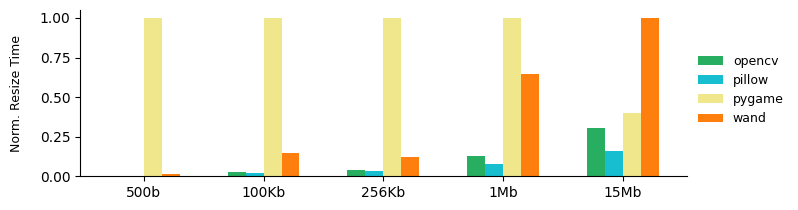
This benchmark downloads an image from cloud storage, resizes it to a thumbnail size, and then uploads the new smaller version of the image. For the experiments, we selected images of different sizes from the image-net dataset used in the EcooFaaS paper. This dataset contains over 1 million images, and from the CDF in Figure 6 we can see that the dominant videos are those whose size is less than, the dominant images in the dataset are those smaller than 0.256 MB (256 KB). We then selected 05 images with sizes of 15 Mb (the largest), 1 Mb, 256 Kb, 100 Kb (the average size of the dataset), and 500 b (the smallest).

  
 *fig.* 1: CDF

We then compared 04 alternative implementations for this benchmark, each with a different python library (opencv, pilow, wand, pygame) to resize the image. For each library with each image, we performed 100 successive executions and the collected metrics were then normalized so that the smallest values were closer to 0 and the largest values closer to 1. The observations and interpretations made after the experiments are as follows:

**Observation 1: Pillow library has the lowest processing times.**

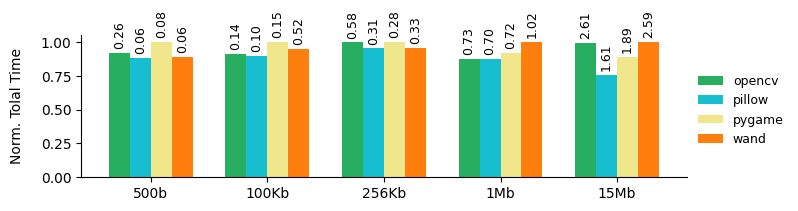
The histogram in Figure 2 shows the average time costs of the resizing operation with the different libraries used for each image size.

*fig.* 2: Average resize time

We observe that, for images of size between 500 b and 1 Mb, the processing time with the Pygame library is significantly higher than the other libraries. However, the gap between Pygame and Wand reduces as the image size increases, to the point where Wand is the slowest library for the 15 Mb image. On the other hand, we keep the lowest processing times with pillow library.

**Observation 2: Pillow library maintains slightly lower total time costs**

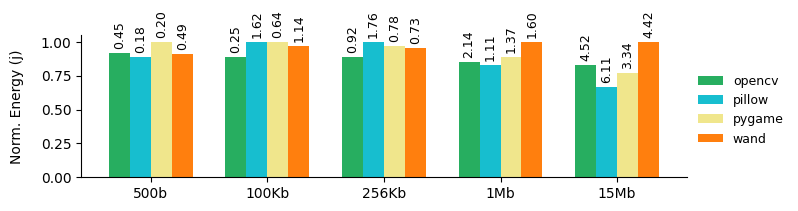
We then compared using the histogram in Figure 3 the average total time cost required to run the benchmark, including the time to resize, download, and upload the image.

  
*fig.* 3: Total average time

At the top of each bar in the histogram is marked the standard deviation of all runs to give an indication of the margin error when experimenting. Looking at the histogram, we see that while the differences in total time costs between the libraries are not very large, Pillow still maintains slightly lower total time costs than the others.

**Observation 3: Pillow and opencv have lower energy consumption**

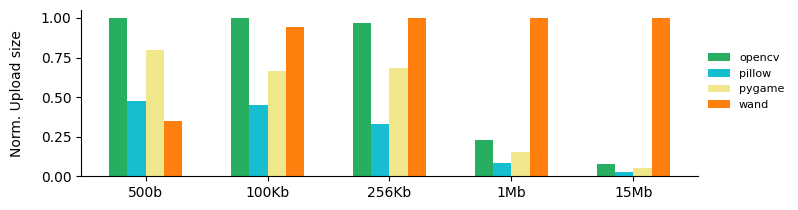
We then looked at the energy consumed when running the benchmark. The histogram in Figure 4 allows us to compare the average energy consumed between the different libraries. We mainly note that the pillow and opencv libraries have lower energy consumption than the others.

  
 *fig.* 4: Average energy consumed

However, it is worth noting that the energy consumption with the Pillow library presents higher standard deviations, indicating that the measurements are more variable and less stable compared to OpenCV.

**Observation 4: OpenCV and Wand produce higher quality final images.**

Finally, we compared the image quality produced by each library based on the final file size. The histogram in Figure 5 shows that, for the most representative images, i.e. those with a size less than 256 Kb, the opencv library produces a higher quality image. For larger images, Wand produces a better result.

  
*fig.* 5: Output image size

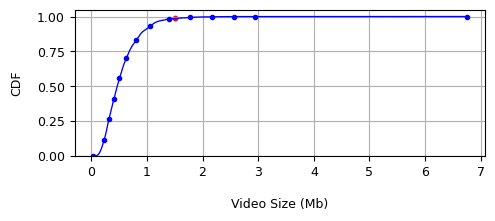
**User side interpretation:** Considering the analysis performed for this benchmark, from the user's point of view we can say that the choice of an alternative will depend on the size of the original image. For example, for images smaller than 256 Kb, the opencv library is the best choice because it produces a higher quality final image, with a total execution time slightly higher than pillow and lower than the others. For larger images, the wand library is the best choice because it produces a considerably higher quality final image with a total execution time slightly higher than the other libraries.

**Provider side interpretation:** On the provider side, the best alternative would be to use the Pillow library, because it allows for all image sizes the lowest total execution time cost and the lowest energy consumption, although highly variable.

**Regarding the motivation, is it interesting ?** Regarding the objectives of the GreenFaaS project, for this first benchmark, we see that there are indeed alternative implementations with different time costs, different energy consumption, and different but acceptable results.

1. **220.video-processing**

This benchmark downloads a video from cloud storage, transforms it into a GIF image, and then returns the result. For the experiments, we selected videos from the [UCF 101 videos](https://www.crcv.ucf.edu/THUMOS14/download.html) dataset also used in the EcoFaaS paper. The dataset contains 13320 videos and from the CDF in Figure 6 we can see that the dominant videos are those whose size is less than 1.5 MB. We then selected 04 videos of sizes 6 MB, 1 MB, 540 Kb, and 36 Kb respectively.

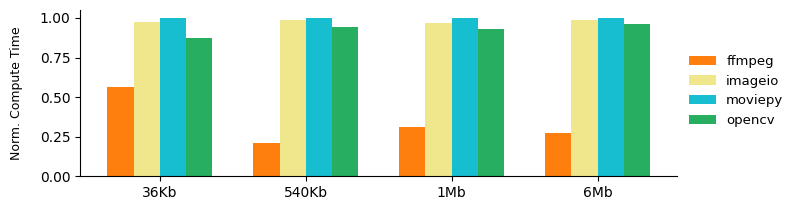


*fig.* 6: CDF

We then implemented and compared 04 alternatives for this benchmark, each with 04 different libraries (ffmpeg, imageio, moviepy, opencv) to transform a video into GIF. For each library as before, with each video, we performed 100 successive executions. The observations and interpretations after the experiments are as follows:

**Observation 1: ffmpeg has the lowest compute time.**

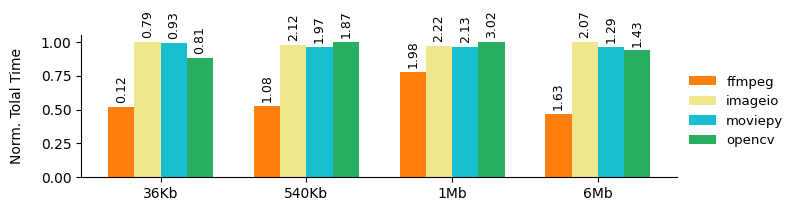
The diagram in Figure 7 shows the average times needed to transform a video into a GIF with each library. We mainly note that ffmpeg has the lowest processing time regardless of the size of the input video. The gap between the other libraries is not very large, but opencv is still quite faster than imageio and moviepy.



*fig.* 7: Average time to transform into GIF

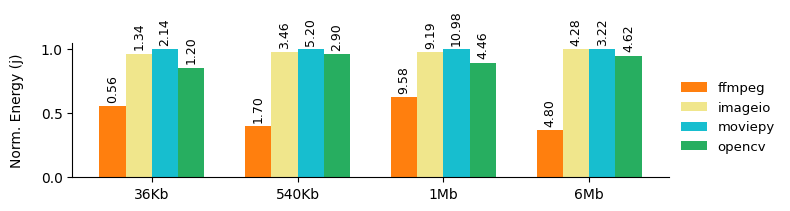
**Observation 2: ffmpeg keeps the lowest total time cost.**

The histogram in Figure 8 shows the average total time costs for the benchmark execution, including the time to transform the video into GIF, download, and upload the result. We mainly observe that ffmpeg keeps a lower total time cost regardless of the input video.

  
*fig.* 8: Total average time

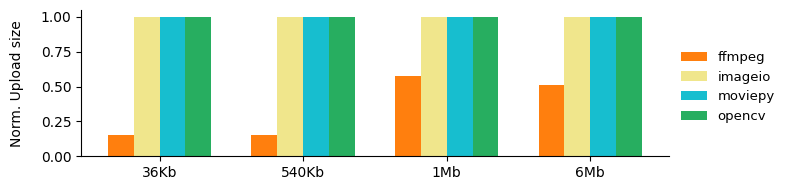
**Observation 3: ffmpeg has the lowest energy consumption.**

The histogram in Figure 9 shows the average energy consumed by each library with different input videos. We mainly note that ffmpeg has the lowest energy consumption regardless of the input video.

  
 *fig.* 9: Average energy consumed

**Observation 4: ffmpeg produces a GIF image of lower quality.**

The histogram in Figure 10 allows us to compare the quality of the final GIF image produced by each library based on the size of the final file. We can see that ffmpeg produces a GIF image with lower quality than the other libraries. Except for ffmpeg, the other libraries produce an output GIF image with the same size and therefore of equal quality.

  
 *fig.* 10: Output GIF image size

**User side interpretation:** Based on the previous observations, from a user perspective, the best alternative for this benchmark would be to use opencv because it produces a higher quality result, with overall time costs closer to ffmpeg.

**Provider side interpretation:** For the provider, the best alternative would be to use ffmpeg because it has the lowest time costs, as well as the lowest energy consumption.

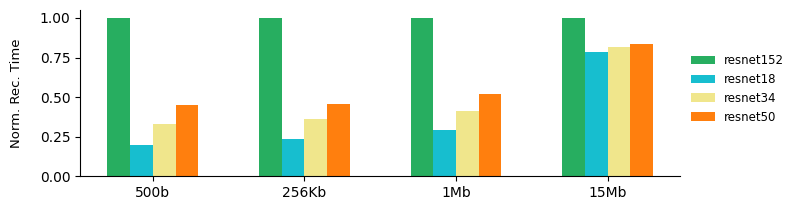
**Regarding motivation, is it interesting?** Regarding the objectives of the GreenFaaS project, we see again that for a specific task there are alternative implementations with different time costs, different energy consumption, and different but acceptable results.

**411.image-recognition**

The benchmark performs an image recognition task. It starts by downloading an image from cloud storage, then submits it as input to a ResNet model, a deep learning model specifically designed for image recognition. The ResNet model then predicts the class of the image and returns the result. For this benchmark, as before, we selected images from the ImageNet dataset. As an alternative implementation, we considered different versions of the ResNet model from the pytorch library, namely ResNet18, ResNet34, ResNet50, ResNet152 denoting respectively different versions of the ResNet model with 18, 34, 50, 152 convolution layers. For each version of the model with each of the images we then performed 100 executions. The observations and interpretations made after the experiments are as follows:

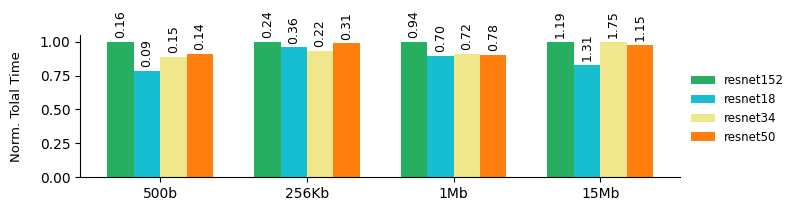
**Observation 1 : ResNet18 a les coûts en temps de traitement le plus bas**

The histogram in Figure 11 shows the average time required for each model to predict the class of each of the images used. We note that ResNet18 has the lowest processing time regardless of the size of the image passed as input.



*fig.* 11: Average prediction time

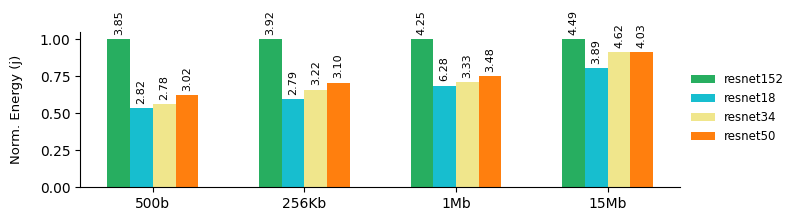
**Observation 2 : Resnet18 conserve les coûts en temps total le plus bas**

The histogram in Figure 12 shows the average total time costs for running the benchmark, including the time to predict the image class, and the image download time. We can see that ResNet18 maintains a slightly lower total time cost overall than the others model.

*fig.* 12: Average total time

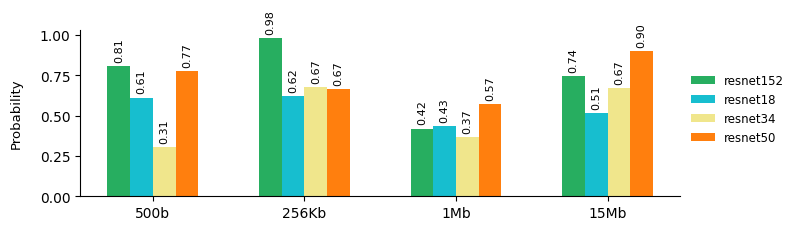
**Observation 3 : Resnet18 a la consommation énergétique la plus basse**

The histogram in Figure 13 shows the average energy consumed by each model with different input images. We mainly note that ResNet18 has the lowest energy consumption for all input images.

 *fig.* 13: Average energy consumption

**Observation 4: Resnet152 and ResNet50 have the most certain predictions**

We then looked at the performance of each variant of the ResNet model by comparing the prediction quality of each of them. The histogram in Figure 14 shows the output probabilities for each model representing the degree of certainty of the model regarding the prediction made for the input image. At the top of each bar of the histogram is marked the value of the probability. We can see that overall, ResNet50 and ResNet152 perform better because they have higher probabilities, and therefore a higher level of confidence.



*fig.* 14: Probability at output

**Provider side interpretation :** Considering the previous observations, and performing an analysis from the user point of view, the best alternative for this benchmark would be to use the ResNet50 model, because with ResNet152 they perform better but ResNet18 has a lower time cost.

**Provider side interpretation :** For the provider, the best choice would be to use ResNet18 because it has the lowest time costs, as well as the lowest energy consumption.

**Regarding motivation, is it interesting ? :** Regarding the objectives of the GreenFaaS project, we can see again that for a specific task there are alternative implementations with different time costs, different energy consumptions, and different but acceptable results satisfying separately the supplier and the user.