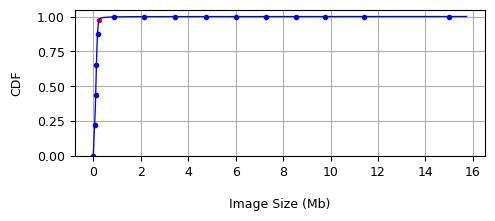
GreenFaaS : La motivation du projet

L’idée du projet GreenFaaS, est de proposer une **nouvelle** **conception des des services FaaS (Function as a Service) permettant de réduire la consommation énergétique lors de l'exécution des fonctions tout en conservant autant que possible les performances (temps d'exécution, qualité du résultat et consommation de ressource).** La première étape du projet est de montrer que pour une tâche (une fonction), il existe des implémentations alternatives dont l'exécution peut produire des résultats différents mais acceptables, avec des consommations énergétiques et consommations de ressources différentes. Pour cela, nous avons commencé par implémenter des benchmarks de l’état de l’art, en concevant et comparant différentes alternatives.

Ce rapport présente l’analyse de l'exécution des benchmarks [SeB’s](https://github.com/spcl/serverless-benchmarks). Les tests ont été réalisés sur la plate-forme **OpenWhisk**, sur une machine hôte équipée d'un processeur à **8 cœurs logiques** sur un seul socket, de **16 Go** de mémoire RAM, le tout fonctionnant sous **Ubuntu 22.04.4 LTS**. Nous avons utilisé le mode “**performance**” du gouverneur fixant la fréquence de tous les cœurs à la fréquence maximale et comme service de stockage distant, nous avons utilisé Amazone S3. Les benchmark ayant déjà été implémenté sont les suivants :

1. **210.thumbnailer**

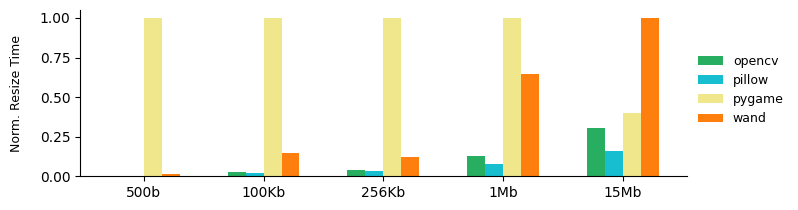
Ce benchmark télécharge une image depuis le stockage cloud, la redimensionne à la taille d'une vignette, upload ensuite la nouvelle version plus petite de l’image. Pour les expérimentations, nous avons sélectionné des images de tailles différentes provenant du jeu de données image-net utilisé dans le papier EcooFaaS. Il contient au total plus 1 millions d’images et grâce au cdf à figure 1, on peut voir que les images dominante dans le jeu de données sont celles dont la taille est inférieure à 0.256 Mb soit 256 Kb. Nous avons alors sélectionné 05 images respectivement de taille 15 Mb (la plus grande), 1 Mb, 256 Kb, 100 Kb (la taille moyenne du jeu de données), et 500 b (la plus petite).

  
 *fig.* 1: CDF

Nous avons par la suite comparé 04 implémentations alternatives pour ce benchmark, chacune avec une bibliothèque python différente (opencv, pilow, wand, pygame) pour redimensionner l’image. Pour chaque bibliothèque avec chacune des images nous avons effectué 100 exécutions successives et les métriques collectés ont ensuite été normalisées de tel sorte que les valeurs les plus petites soient plus proches de 0 et les plus grandes plus proches de 1. Les observations et interprétations faites après les expérimentations sont les suivantes :

**Observation 1**: **La bibliothèque pillow a des temps de traitement les plus bas**

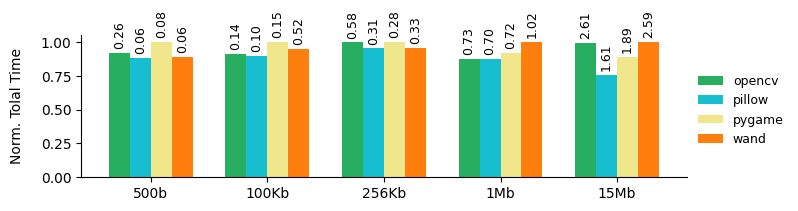
L'histogramme à la figure 2 présente les coûts moyens en temps de l’opération de redimensionnement avec les différentes bibliothèques utilisées pour chaque taille image.

  
*fig.* 2: Temps moyen de redimensionnement

On observe que, pour des images de taille comprise entre 500 b et 1 Mb, le temps de traitement avec la bibliothèque Pygame est largement supérieur par rapport aux autres bibliothèques. Toutefois, l'écart entre Pygame et Wand se réduit à mesure que la taille de l’image augmente, au point où Wand est la bibliothèque la plus lente pour l’image de 15 Mb. On conserve, en revanche, des temps de traitement les plus bas avec pillow.

**Observation 2: La bibliothèque pillow conservent des coûts en temps total légèrement inférieurs aux autres bibliothèques**

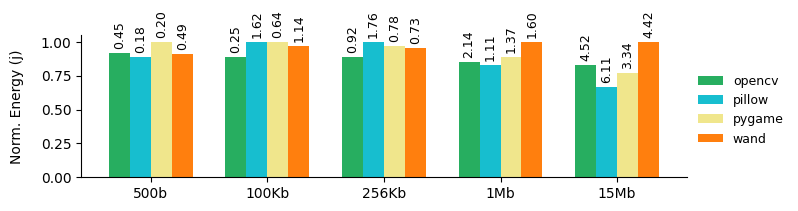
Nous avons ensuite comparé grâce à l’histogramme de la figure 3 le coût moyen total en temps nécessaire pour l’exécution du benchmark, incluant le temps de redimensionnement, de téléchargement, et retour de l’image.

  
*fig.* 3: Temps moyen total

Au sommet de chaque barre de l'histogramme est marqué l'écart type de l'ensemble des exécutions pour avoir une indication sur la marge d’erreur lors des expérimentations. En observant l'histogramme, nous constatons que, même si les différences de coûts en temps total entre les bibliothèques ne sont pas très grandes, Pillow conserve néanmoins des coûts total en temps légèrement inférieurs par rapport aux autres.

**Observation 3: Pillow et opencv ont des consommations énergétiques plus basses que les autres bibliothèques.**

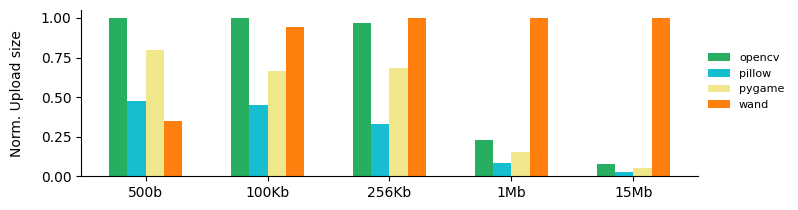
Nous nous sommes ensuite intéressés à l'énergie consommée lors de l'exécution du benchmark. L'histogramme à la figure 4 permet de comparer l'énergie moyenne consommée entre les différentes bibliothèques . On relève principalement que, les bibliothèques pillow et opencv ont une consommations énergétiques plus basses que les autres.

  
 *fig.* 4: Énergie moyenne consommée

On note tout de même le fait que, la consommation énergétique avec la bibliothèque pillow présente des écarts types plus élevés, ce qui indique que les mesures sont plus variables et moins stables par rapport à opencv.

**Observation 4** : **OpenCV et Wand produisent des images finales de meilleure qualité**.

Pour finir nous avons comparé la qualité de l'image produite par chaque bibliothèque en se basant sur la taille du fichier final. L’histogramme à la figure 5 ci-dessous montre que, Pour des images les plus représentatives, c'est-à-dire celles dont la taille est inférieur à 256 Kb, la bibliothèque opencv produit une image de meilleure qualité. Pour des images de taille plus grande, Wand produit un meilleur résultat.

  
*fig.* 5: Taille de l’image en sortie

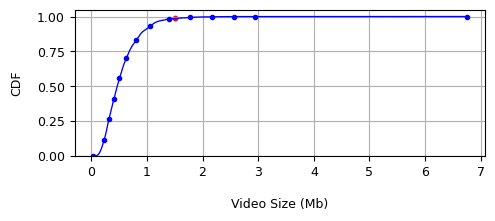
**Interprétation côté utilisateur:** En tenant compte des analyses effectuées pour ce benchmark, du point de vue de l’utilisateur on peut dire que le choix d'une alternative dépendra de la taille de l’image d’origine. Par exemple, pour des images de tailles inférieures à 256 Kb, la bibliothèque opencv se présente comme le meilleur choix car elle produit une image finale de meilleure qualité, avec un temps d’exécution total légèrement supérieur à pillow et plus bas que les autres. Pour des images de taille plus grande, la bibliothèque wand est le meilleur choix car elle produit une image finale considérablement de meilleure qualité avec un temps d’exécution total légèrement supérieur aux autres bibliothèques.

**Interprétation côté fournisseur:** Du côté fournisseurs, la meilleure alternative serait d'utiliser la bibliothèque Pillow, car elle permet d’avoir pour toutes tailles d'image, le coût total en temps d’exécution le plus bas et des consommations énergétiques les plus basses bien que très variables.

**Concernant la motivation, est-ce intéressant ?** En ce qui concerne les objectifs du projet GreenFaaS, pour ce premier benchmark, on constate qu’il existe effectivement des implémentations alternatives avec des coûts en temps différents, des consommations énergétiques différentes, et des résultats différents mais acceptables.

1. **220.video-processing**

Ce benchmark télécharge une vidéo depuis le stockage cloud, la transforme en image au format GIF, et retourne ensuite le résultat. Pour les expérimentations, nous avons sélectionné des vidéos provenant du jeu de données [UCF 101 videos](https://www.crcv.ucf.edu/THUMOS14/download.html) également utilisé dans le papier EcoFaaS. Le jeu de données contient 13320 vidéos et à partir du CDF à la figure 6 on constate que les vidéos dominante sont celles dont la taille est inférieure à 1.5 Mo. Nous avons alors sélectionné 04 vidéos respectivement de tailles 6 Mo, 1 Mo, 540 Kb, et 36 Kb.

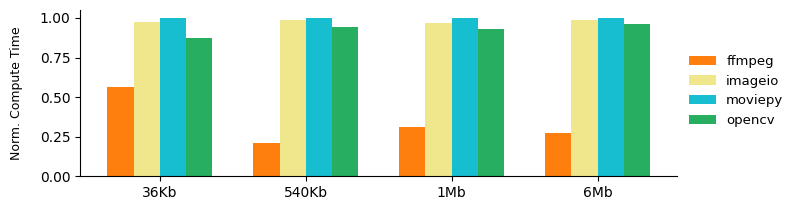


*fig.* 6: CDF

Nous avons par la suite implémenté et comparé 04 alternatives pour ce benchmark, chacune avec 04 bibliothèques différentes (ffmpeg, imageio, moviepy, opencv) pour transformer une vidéo en GIF. Pour chaque bibliothèque comme précédemment, avec chaque vidéo, nous avons effectué 100 exécutions successives. Les observations et interprétations faites après les expérimentations sont les suivantes :

**Observation 1 : ffmpeg a le temps de traitement le plus bas.**

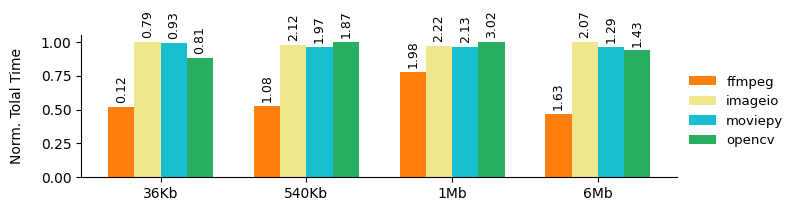
Le diagramme à la figure 7 présente les temps moyens nécessaires pour transformer une vidéo en GIF avec chaque bibliothèque. On note principalement que, ffmpeg a le temps de traitement le plus bas peu importe la taille de la vidéo en entrée. L'écart entre les autres bibliothèques n'est pas très grand, mais opencv est tout de même assez plus rapide que imageio et moviepy.



*fig.* 7: Temps moyen de transformation en GIF

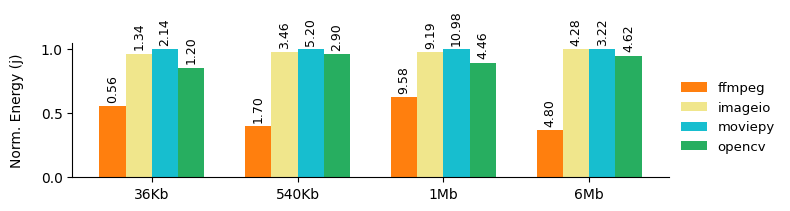
**Observation 2: ffmpeg conserve le coût en temps total le plus bas.**

L 'histogramme à la figure 8 présente les coûts moyens total en temps pour l’exécution du benchmark, incluant le temps de transformation de la vidéo en GIF, de téléchargement, et retour du résultat. On constate principalement que ffmpeg conserve un coût total en temps plus bas peu importe la vidéo en entrée.

  
*fig.* 8: Temps moyen total

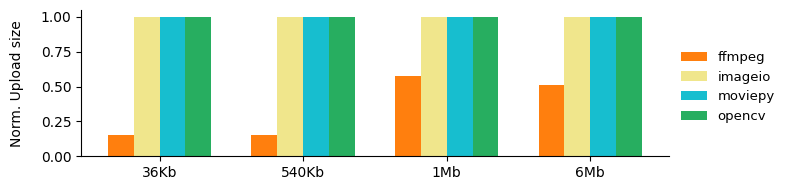
**Observation 3: ffmpeg a des consommations énergétiques les plus basses.**

L'histogramme à la figure 9 présente l'énergie moyenne consommée par chaque bibliothèque avec différente vidéo en entrée. On note principalement que ffmpeg a la consommation énergétique la plus basse quelque soit la vidéo en entrée.

  
 *fig.* 9: Énergie moyenne consommée

**Observation 4: ffmpeg produit une image GIF de moins bonne qualité.**

L 'histogramme à la figure 10 permet de comparer la qualité de l'image GIF finale produite par chaque bibliothèque en se basant sur la taille du fichier final. On constate que ffmpeg produit une image GIF de moins bonne qualité que les autres bibliothèques. Excepté ffmpeg les autres bibliothèques produisent en sortie une image GIF de même tailles et donc de qualité égale.

  
 *fig.* 10: Taille de l’image GIF en sortie

**Interprétation côté utilisateur :** Sous la base des observations précédentes, du point de vue utilisateur, la meilleure alternative pour ce benchmark serait d'utiliser opencv car il produit un résultat de meilleure qualité, avec des coûts en temps globalement plus proches de ffmpeg.

**Interprétation côté fournisseur :** Pour le fournisseur, la meilleure alternative serait d'utiliser ffmpeg car il a des coûts en temps les plus bas, ainsi que des consommations énergétiques les plus basses.

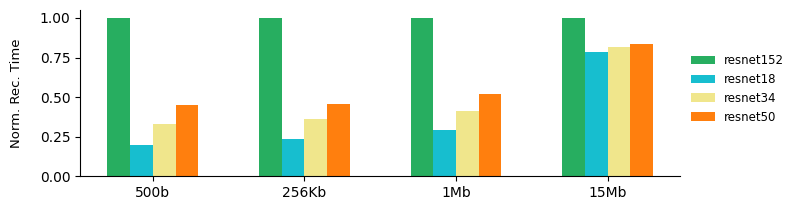
**Concernant la motivation, est-ce intéressant ?** En ce qui concerne les objectifs du projet GreenFaaS, on constate là encore qu’il existe pour une tâche précise des implémentations alternatives avec des coûts en temps différents, des consommations énergétiques différentes, et des résultats différents mais acceptables.

1. **411.image-recognition**

Le benchmark exécute une tâche de reconnaissance d'image. Il commence par télécharger une image depuis le stockage cloud, puis la soumet en entrée à un modèle ResNet, un modèle de deep learning spécialement conçu pour la reconnaissance d'images. Le modèle ResNet prédit alors la classe de l'image et renvoie le résultat. Pour ce benchmark, comme précédemment, nous avons sélectionné des images provenant du jeu de données ImageNet. Comme implémentation alternative, nous avons considéré différentes versions du modèle ResNet provenant de la bibliothèque pytorch, à savoir ResNet18, ResNet34, ResNet50, ResNet152 désignant respectivement des versions différentes du modèle ResNet avec 18, 34, 50, 152 couches de convolutions. Pour chaque version du modèle avec chacune des images nous avons ensuite effectué 100 exécutions. Les observations et interprétations faites après les expérimentations sont les suivantes :

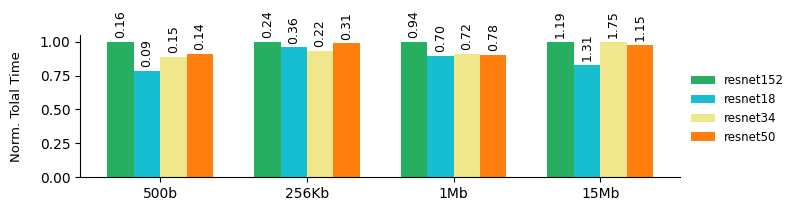
**Observation 1 : ResNet18 a les coûts en temps de traitement le plus bas**

L'histogramme à la figure 11 présente le temps moyen nécessaire à chaque modèle pour prédire la classe de chacune des images utilisées. On note que ResNet18 a le temps de traitement le plus bas quelque soit la taille de l’image passée en entrée.



*fig.* 11: Temps moyen de prédiction

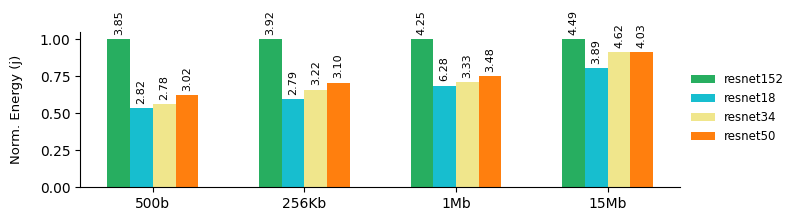
**Observation 2 : Resnet18 conserve les coûts en temps total le plus bas**

L'histogramme à la figure 12 présente les coûts moyens totaux en temps pour l’exécution du benchmark, incluant le temps pour prédire la classe de l’image, et le temps de téléchargement de l’image. On constate que ResNet18 conserve globalement un coût total en temps légèrement plus bas que les autres.

*fig.* 12: Temps moyen de total

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

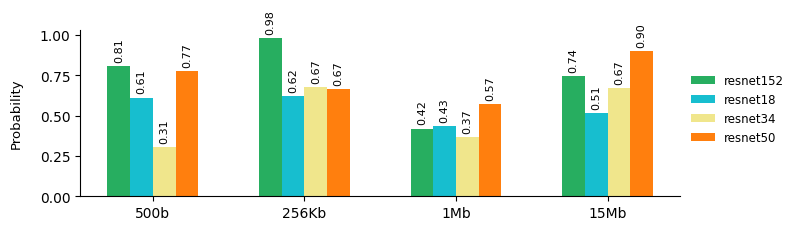
L'histogramme à la figure 13 présente l'énergie moyenne consommée par chaque modèle avec des images de différentes tailles en entrée. On note principalement que ResNet18 a la consommation énergétique la plus basse pour toutes les images passées en entrée.



*fig.* 13: Consommation énergétique moyenne

**Observation 4 : Resnet152 et ResNet50 ont les prédictions les plus certaines**

Nous nous sommes ensuite intéressés aux performances de chaque variante du modèle ResNet en comparant la qualité de prédiction par chacune d’elle. L’histogramme à la figure 14 présente les probabilités en sortie pour chaque modèle représentant le degré de certitude du modèle quant à la prédiction faite pour l’image en entrée. Au sommet de chaque barre de l'histogramme est marqué la valeur de la probabilité. On constate que dans l’ensemble, ResNet50 et ResNet152 sont plus performant car ils ont des probabilités les plus élevés, et donc un niveau de confiance plus grand.



*fig.* 14: Probabilité en sortie

**Interprétation côté utilisateur :** En considérant les observations précédentes, et en effectuant une analyse du point de vue utilisateur, la meilleure alternative pour ce benchmark serait d'utiliser le modèle ResNet50, car avec ResNet152 ils sont plus performant mais ResNet18 a un coût en temps plus bas.

**Interprétation côté fournisseur :** Pour le fournisseur, le meilleur choix serait d'utiliser ResNet18 car il a des coûts en temps les plus bas, ainsi que des consommations énergétiques les plus basses.

**Concernant la motivation :** En ce qui concerne les objectifs du projet GreenFaaS, on constate là encore qu’il existe pour une tâche précise des implémentations alternatives avec des coûts en temps différents, des consommations énergétiques différentes, et des résultats différents mais acceptables satisfaisant séparément le fournisseur et l’utilisateur.