Benchmark of Cyfhel

Links:

https://github.com/remyauda/Cyfhel

https://github.com/shaih/HElib

abstract

Cyfhel is a library that is an abstraction of Helib in C ++ for homomorphic encryption. This report is a study measuring the performance of the Cyfhel library for each of the homomorphic operations allowed by the library.

Table des matières

1 Introduction

In this part of the document, we present the results of the benchmark performed on the library Cyfhel. It is also a good overview of the overall performance of the homomorphic encryption.

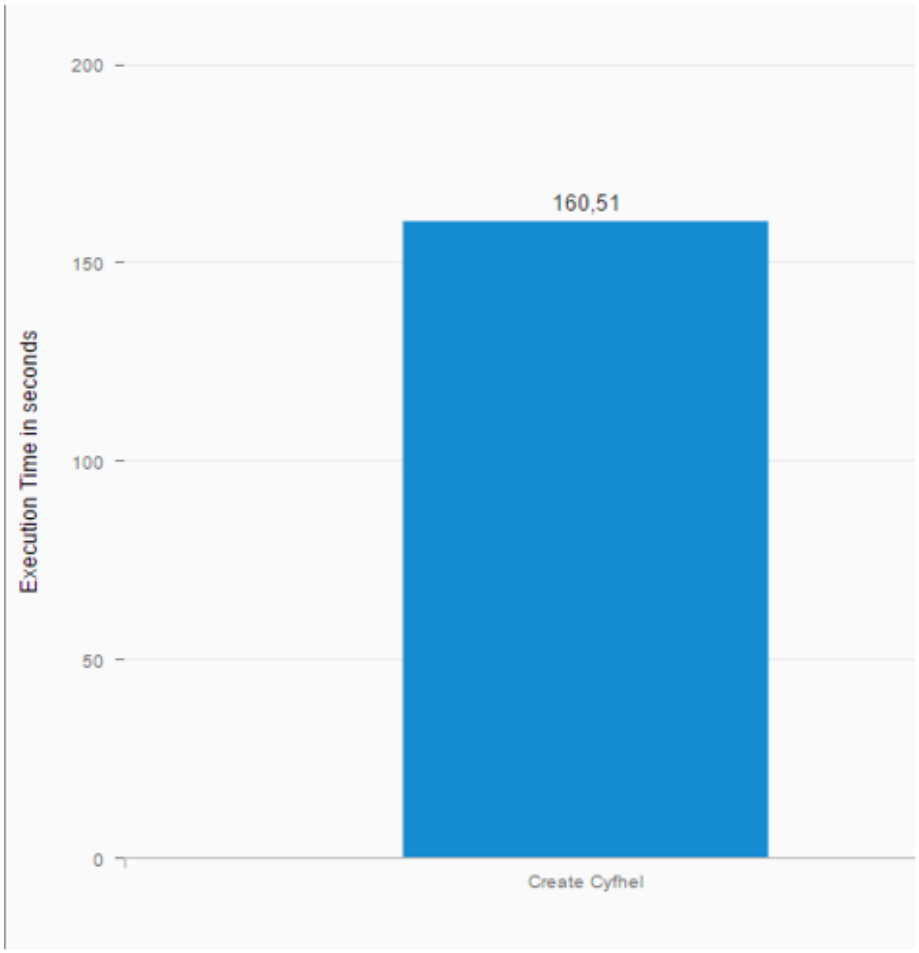
2 Study of encryption and decryption performance for Cyfhel

On the following graph, you can see the time it takes to create an object Cyfhel. Create an object Cyfhel is very long as you can see because there is the creation of the context and the generation of the public and private keys.

The benchmark has been done with the following parameters:

```
long p = 2, long r = 32, long c = 2, long d = 1, long sec = 128, long w = 64, long L = 40,
long m = -1, long const& R = 3, long const& s = 0, vector<long> const& gens =
vector<long>(), vector<long> const& ords = vector<long>().
```

```
We have therefore p^r = 4 294 967 296.
```



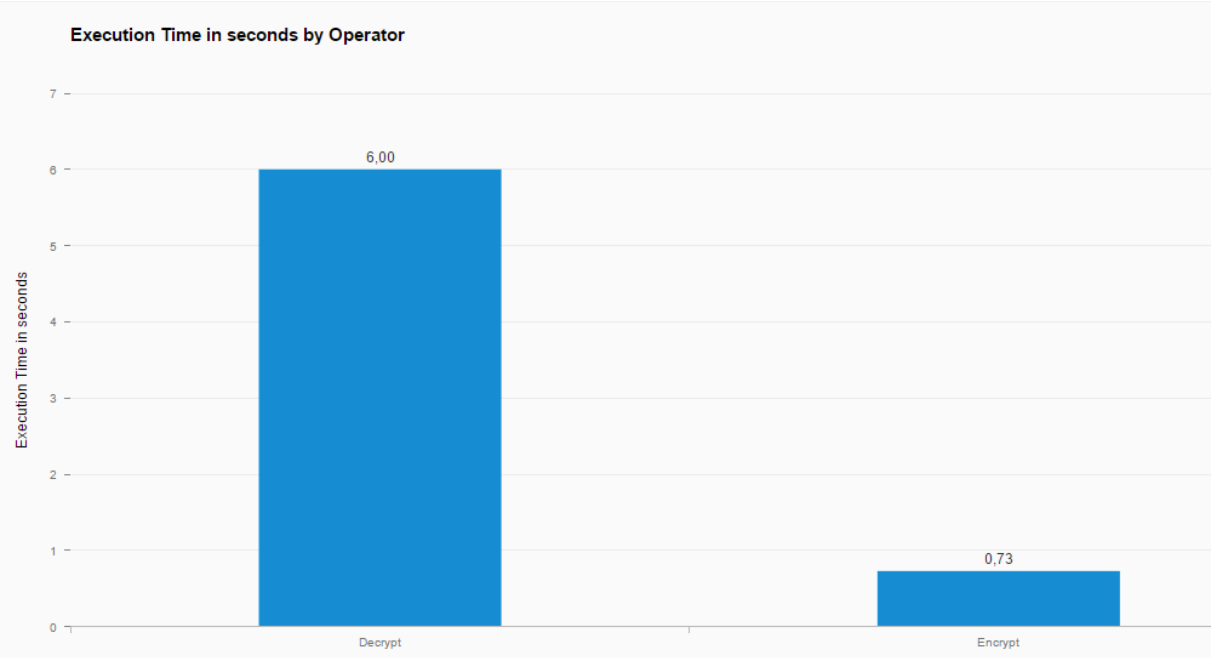
2.1 *Parameters of the key generation used during the benchmark*

As we say above, all the benchmark is done using the following parameters:

```
long p = 2, long r = 32, long c = 2, long d = 1, long sec = 128, long w = 64, long L = 40,
long m = -1, long const& R = 3, long const& s = 0, vector<long> const& gens =
vector<long>(), vector<long> const& ords = vector<long>().
```

We have therefore p^r = 4 294 967 296.

2.2 *Benchmark of encryption and decryption of Cyfhel*



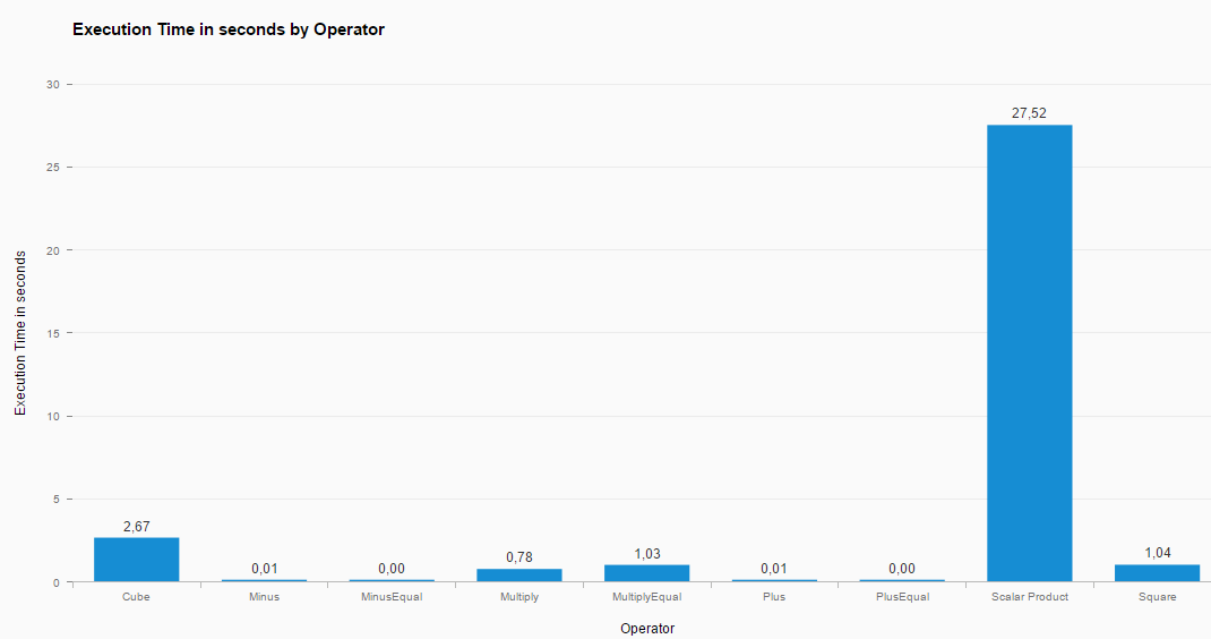
As you can see, the decryption is very long compared to the encryption. But you have to note that this is due to additional treatments that we have done in the implementation of Cyfhel to facilitate the life of the end-user that will use the library.But these treatments can be optimize, so in fact, even if this graph shows that the decryption is longer than encryption, in future version of Cyfhel, the decryption function will be improved to have more or less the same performance as the encryption.

Indeed, if we measure only the time of the decryption without our additional treatments, we obtain also a decryption time around 0.75 seconds.

So, to conclude we can say that the homomorphic encryption and decryption take less than a second to be performed (around 0.75 seconds) and that the decryption function in Cyfhel could be optimize to offer the same easy to use for the final user but with good performances.

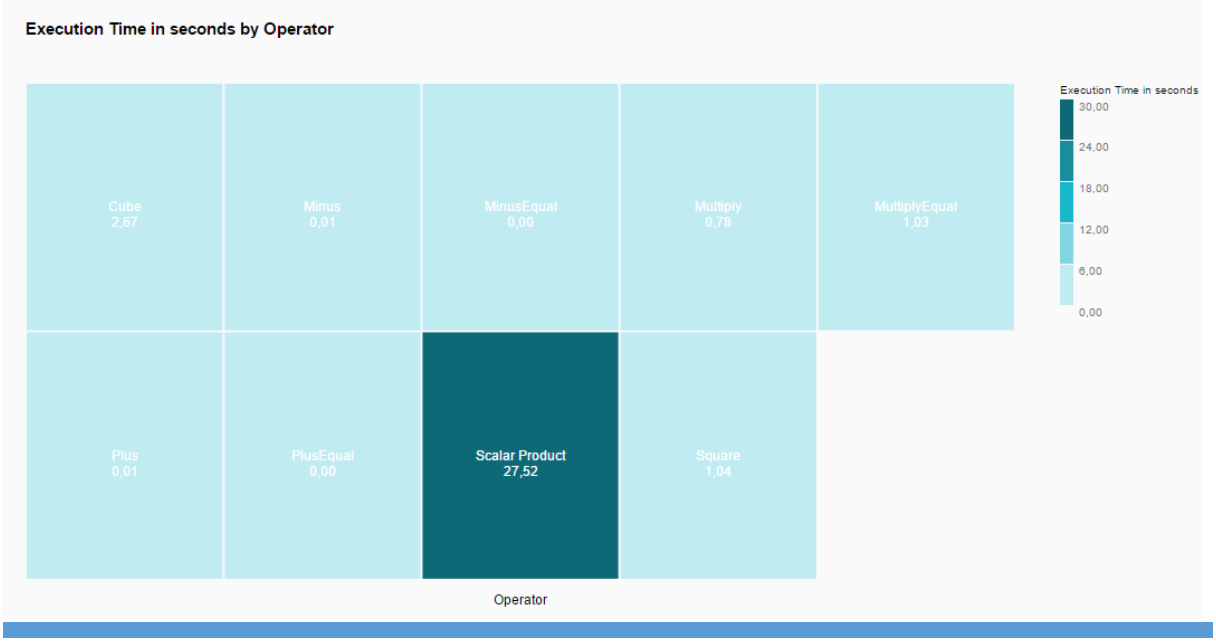
3 Study of basic operations performance for Cyfhel

We have also performed some benchmarking of the basic homormorphic operations like the addition, substraction, multiplyBy and the scalar product between encrypted vectors.



We can see that the following operations: += and -= are very fast operations. The + and – operations are a little slower but remain very fast too (around 0.01 seconds). As expected, the multiply operation is less fast but still fast: * is around 0.78 seconds in average and *= around one second. The square take around 1 second too while the cube operation take around 2.5 seconds to be performed.

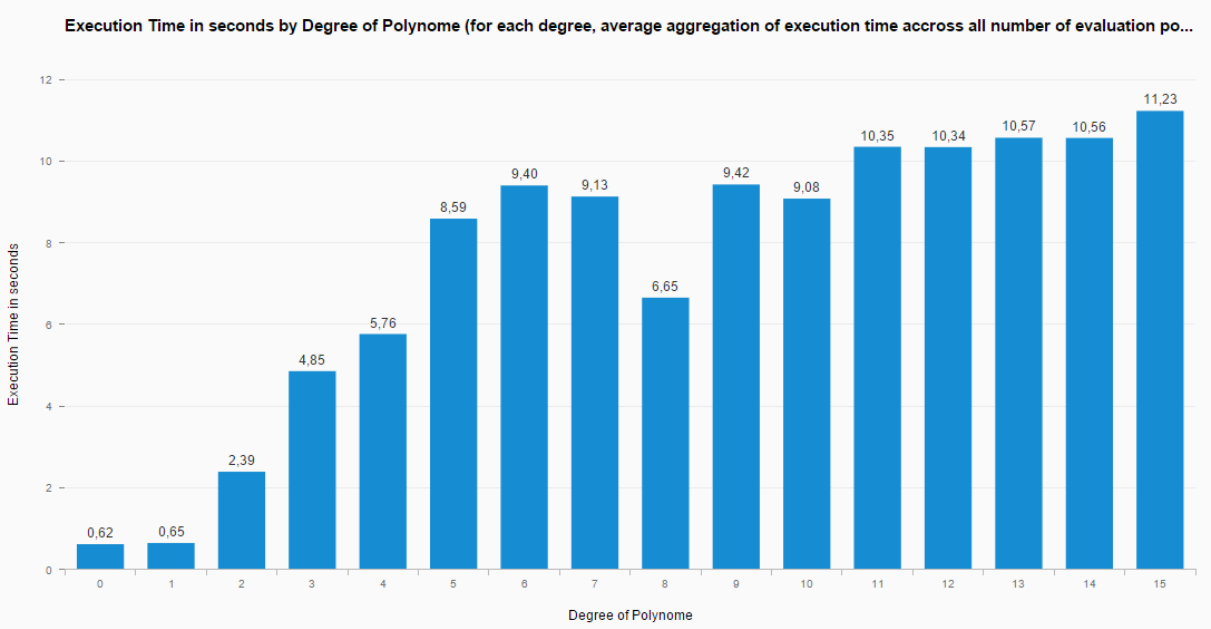
Then, we can see that there is an issue on the performance for the scalar product. Of course, we expect that the scalar product takes more time than a basic operation as it required more operations but here it takes too long. It is due that HElib doesn't implement natively the scalar product so we were obliged to implement this function by hand. Here we can see that the performance are not the ones expected so we will improved the performance of this function in future version of Cyfhel.



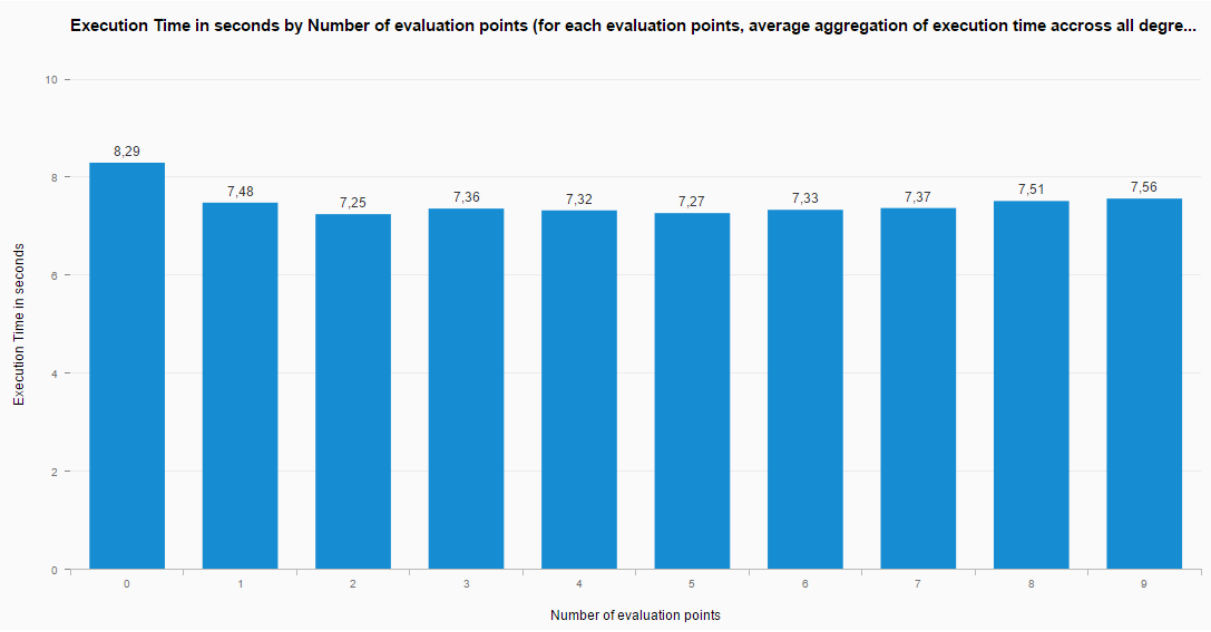
4 Study of polynomial evaluation performance for Cyfhel

The study of the polynomial evaluation is very important for our project as the activation function of the convolutional neural network will be some polynomial approximation of the sigmoid like ReLu.

To perform this benchmark, we have done n times the evaluation of a vector of size N by a polynome of degree d and compute the average amount of time required to perform the polynomial evaluation.

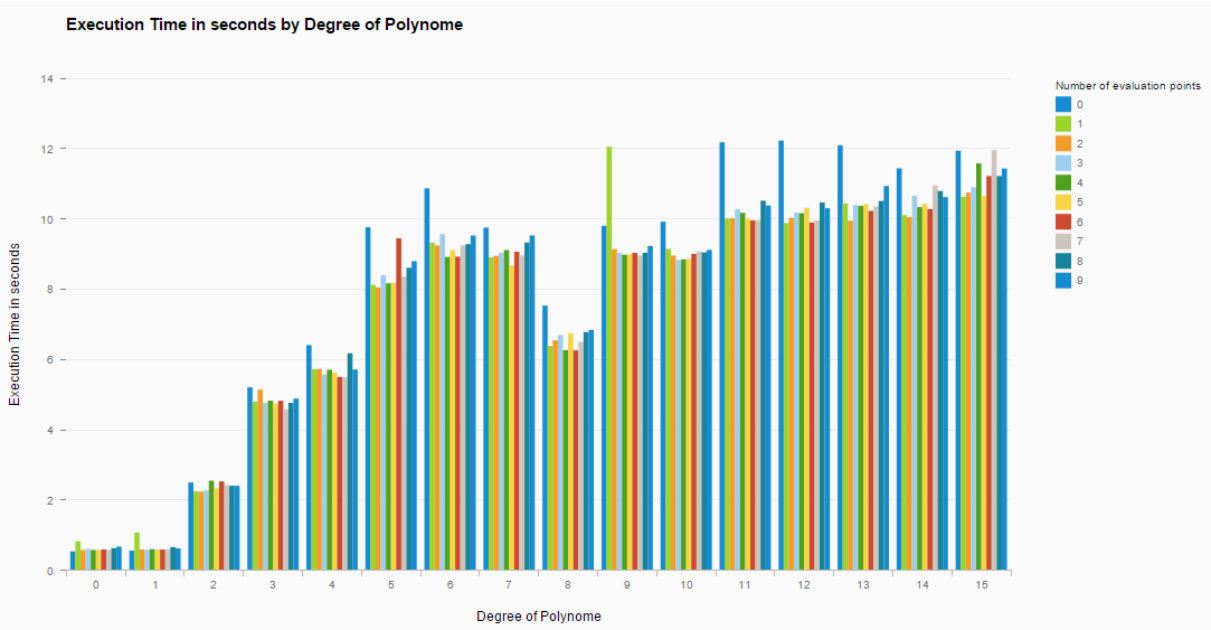


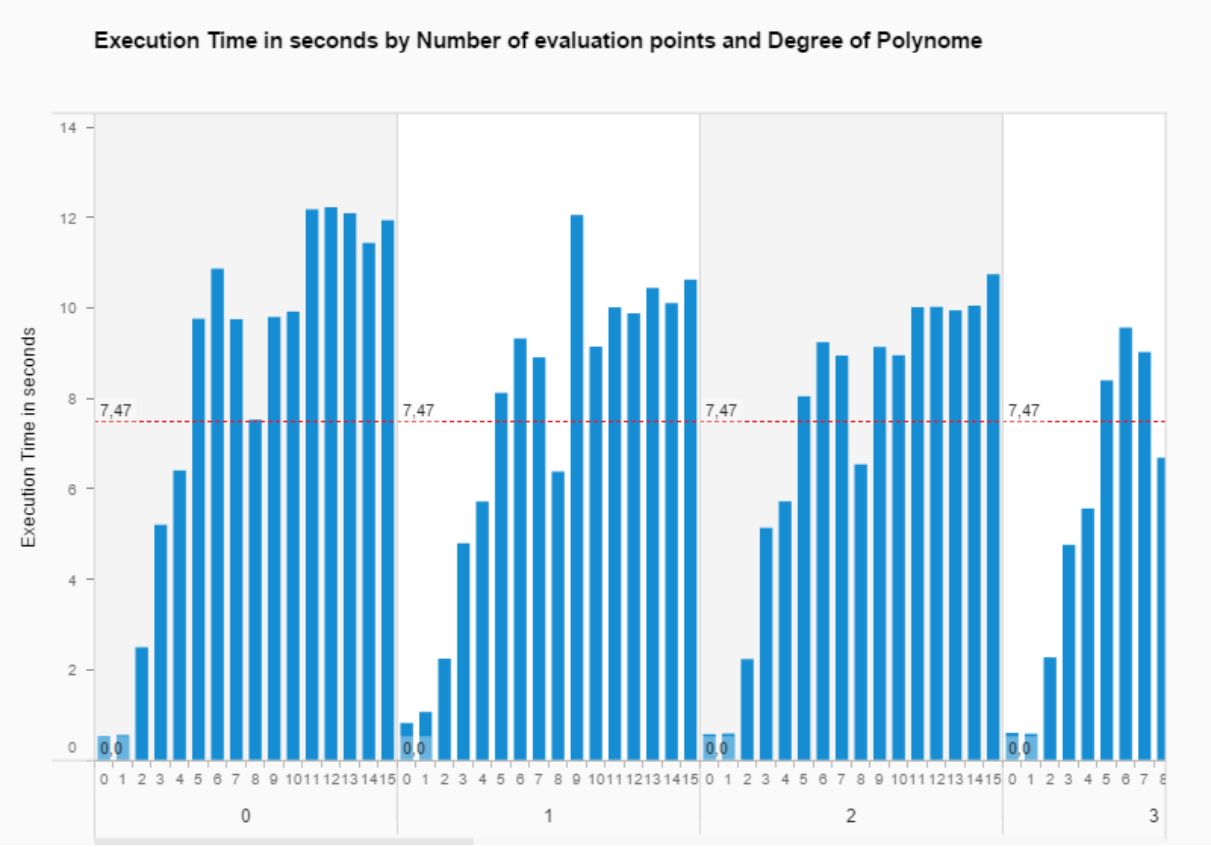
We can see here that the more the degree of the polynome increases, the more the polynomial evaluation will take time. There is an issue in that trend for degree 8 but there are a lot of factors for this inconsistency (less jobs performed by the CPU at this time etc…). Moreover, as we see in the following graph:

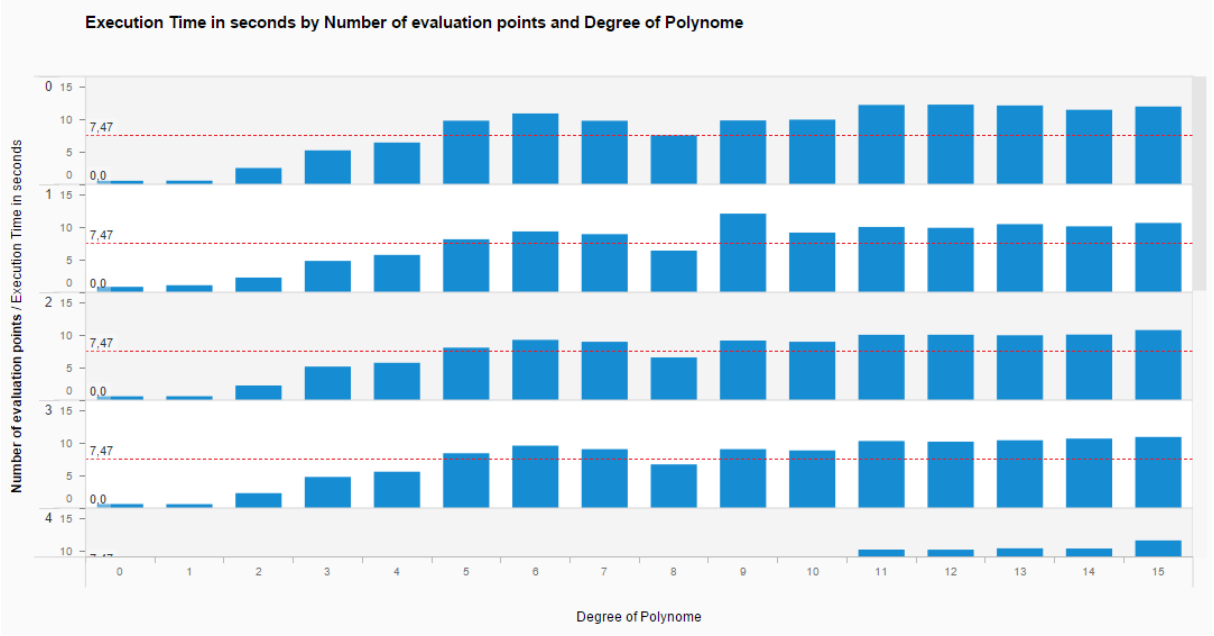


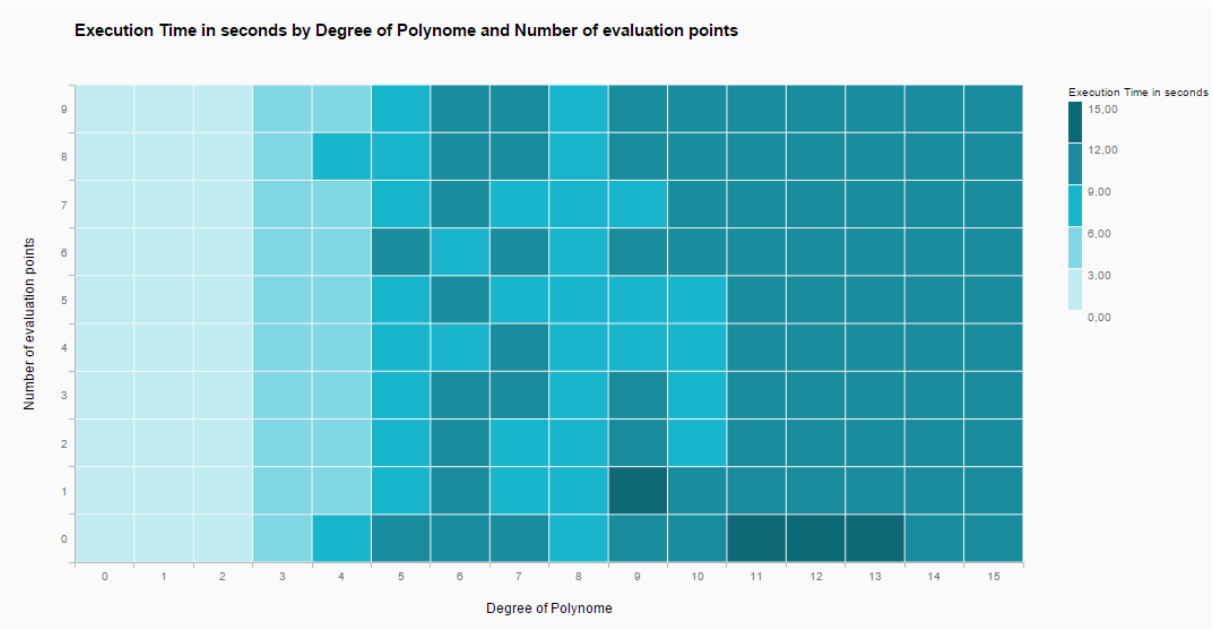
The more evaluation points, the slower the evaluation will be. We can note that it is not true when there are very few evaluation points (for instance 1 evaluation point). We have to investigate deeper to understand if it is the design of HElib that cause this fact or if it is due to the design of the test. However, we certainly could improve Cyfhel in the future version to go faster when evaluate only very few evaluation points.

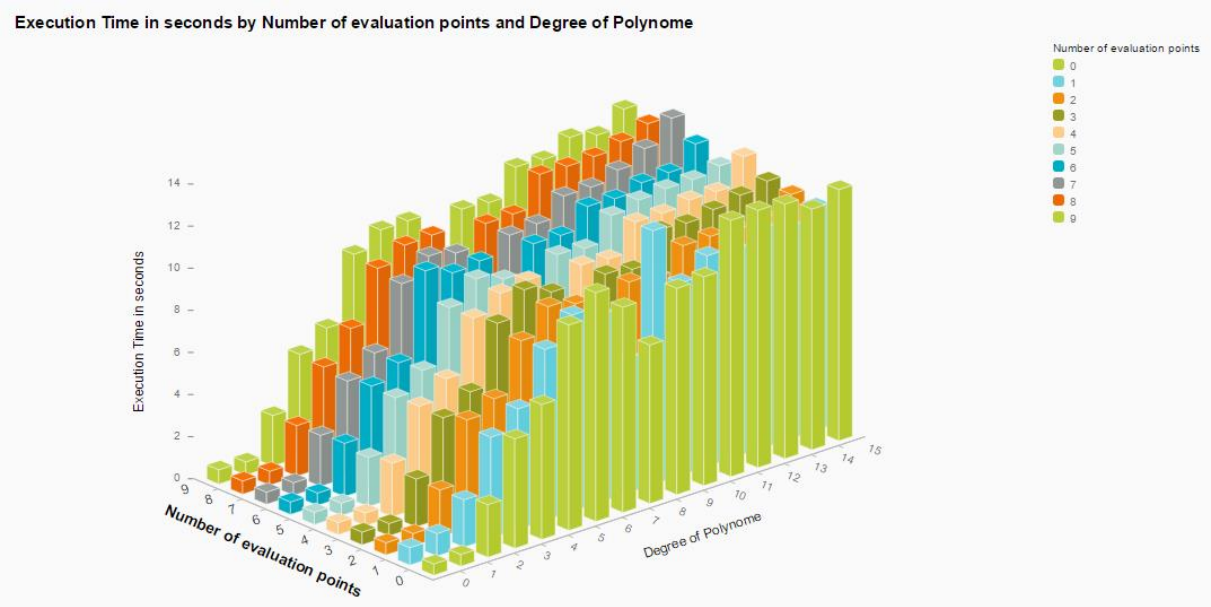
Here is a graph with all the parameters and variables:






Execution Time in seconds by Number of evaluation points and Degree of Polynome
Number of evaluation points / Execution Time in seconds
7,47
0,0
Degree of Polynome


Execution Time in seconds by Degree of Polynome and Number of evaluation points
Number of evaluation points
Degree of Polynome
Execution Time in seconds
15,00
12,00
9,00
6,00
3,00
0,00



5 Conclusion

To conclude, we have seen that the creation of the object Cyfhel is very long as it contains the generation of the context and the generation of the keys. However, it is not really a problem in our project as the generation of the keys, the encryption and the decryption are done by the client himself and not by the Convolutional Neural Network provider. Thus, the important part for our project are the duration of the operations and the polynomial evaluation. As we have seen, most of the operator are quite fast and some can be improve like the Cube and the scalar product operation.

For the polynomial evaluation, we have seen that the number of evaluation points influences the time of evaluation but at a very little level compared to the increase of the degree. Still, we obtain a polynomial evaluation of 15 evaluation points with a polynome of degree 15 in 15 seconds which can be acceptable.

We can note that we can improve also the performance by parallelize the operations.

Moreover, this benchmark has been perform on Cyfhel (a C++ library) but the magnitude of the performance remains true for Pyfhel except that each operation will take longer due to Python.