# Franco-Arabic Conversion 101 Summary

### people of Egypt have developed and adopted a new way to write Arabic in English figures and numbers. Recently Facebook started banning and restricting accounts over racist comments or inappropriate usage of words. But till this current moment, Franco-Arabic is not dominated by Facebook algorithms.

### this notebook (which is nothing more than a playground) aims for the word conversion from I.E ("al-Salam" to "السلام"). In order to do this, certain things should be taking into consideration such as (vowels conversion, spaces between articles and words, similarly pronounced letters…).

### the output of this functionality is of very low readability, which means it is not targeting end-users or humans, but a machine that can interpret a meaningless Arabic expression or word, and then add it to the training Arabic glossary and try to match it to its original word. More Ideas and processing techniques were applied in my original code, but they were removed for the sake of simplicity, that is the reason Emojis are in an odd unfamiliar format in addition to some rarely used numbers and special characters.

### various ways of parallelism were used in this notebook to enhance the computation performance and runtime. (lambda, Multiprocessing, Pandarallel, Multiprocessing and pool, Swifter). There was no significant difference found in performance amongst all these threading techniques. On the contrary, a regular apply method with a lambda function was the easiest to apply and the fastest.

### For credibility I executed this notebook eleven times to make sure that every runtime result is not just a matter of randomness, all the techniques used in the notebook yielded a runtime of approximately two minutes. The runtime seconds difference was in favor of the lambda and apply technique, followed by the Multiprocessing and pool and at the end came Swifter and Pandarallel with a difference range 98 to 122 seconds.

### I do think that two minutes runtime is probably the best performance I can get running the code on a single node. In other words, the parallelism techniques could not do better because there is no better to be done in this case. And I support this claim by the fact that I failed to apply the same functionality using a simple for loop, for some reason the kernel died every single time I executed the code.

### Overall, the result of parallelism is so appealing especially when it comes to a big textual dataset that requires looping over every single character. But eventually too much is too much. At some point, a single node will not be able to give the required performance and runtime, and so the single node should be replaced by a cluster of nodes.