Deep learning techniques for the detection and classification of vulnerabilities in SMART CONTRACTS deployed on the ETHERUM mainnet

Project for the course of Blockchain & Cryptocurrencies

By Gianluca Di Tuccio and Lorenzo Orsini

Starting point: "*On the Use of Deep Neural Networks for Security Vulnerabilities Detection in Smart Contracts*"

TABLE II
MODELS' RESULTS ON THE VALIDATION SET

Model name	Accuracy	Micro F1
ResNet1D	0.7353	0.8381
ResNet	0.6841	0.7928
Inception	0.6988	0.8015
LSTM Baseline	0.6934	0.7953

- **Good performances by taking the bytecodes as input;**
- **It takes only the first bytecodes as input (without trying different input portions)**
- **Self attention model wasn't explored;**

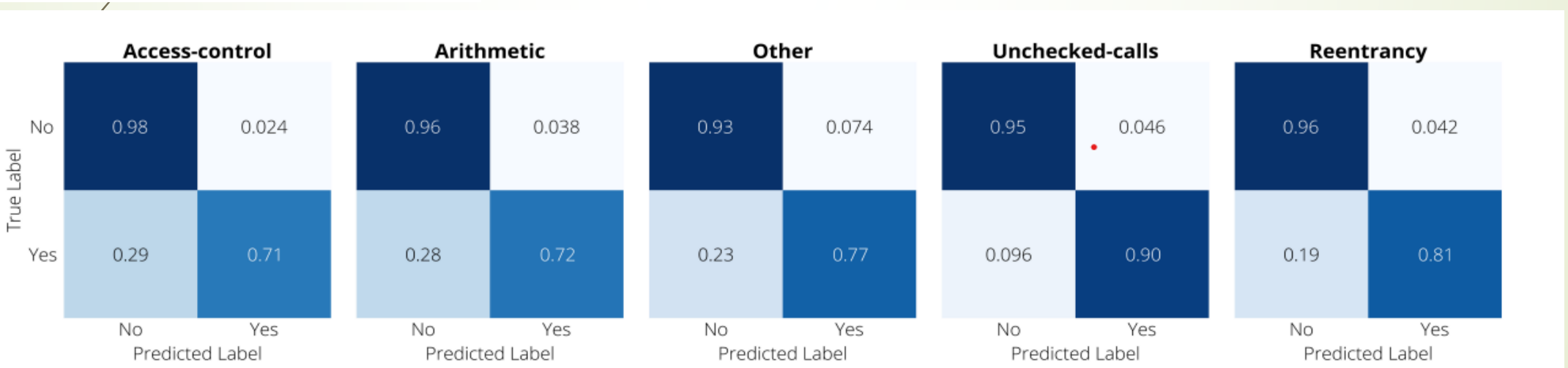
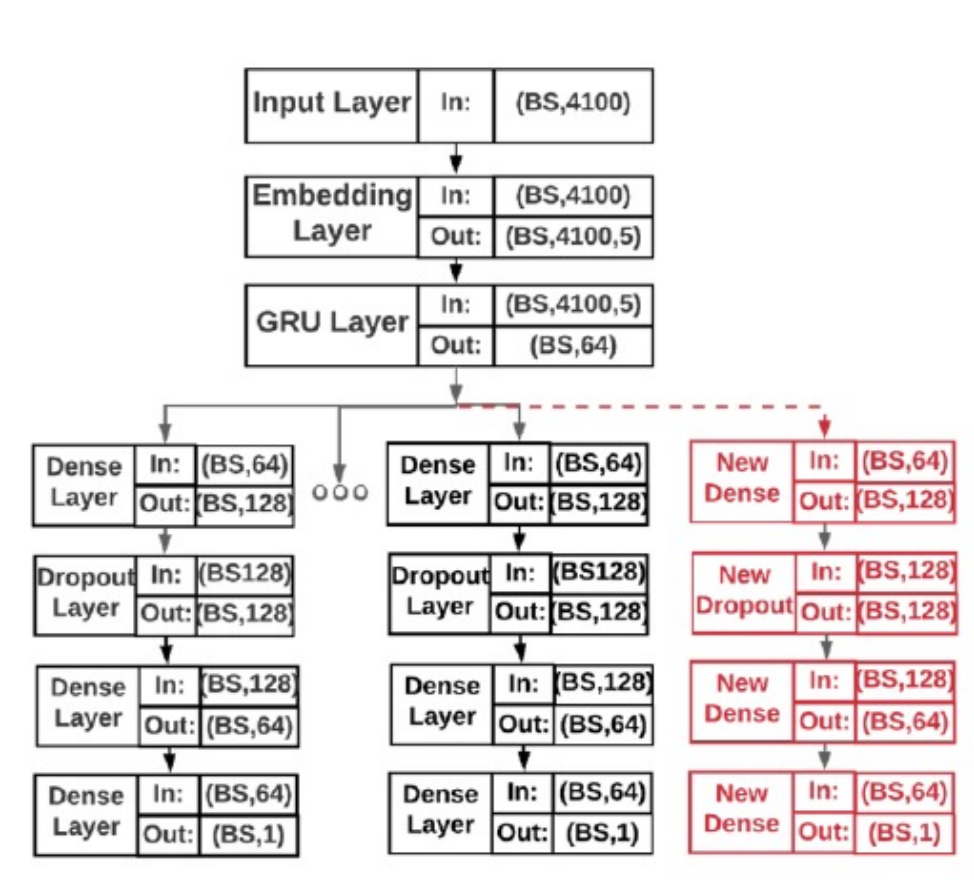


Fig. 5. Per-class confusion matrices obtained by ResNet1D on the test set

Starting point: "*ESCORT: Ethereum Smart COntRacTs Vulnerability Detection using Deep Neural Network and Transfer Learning*"



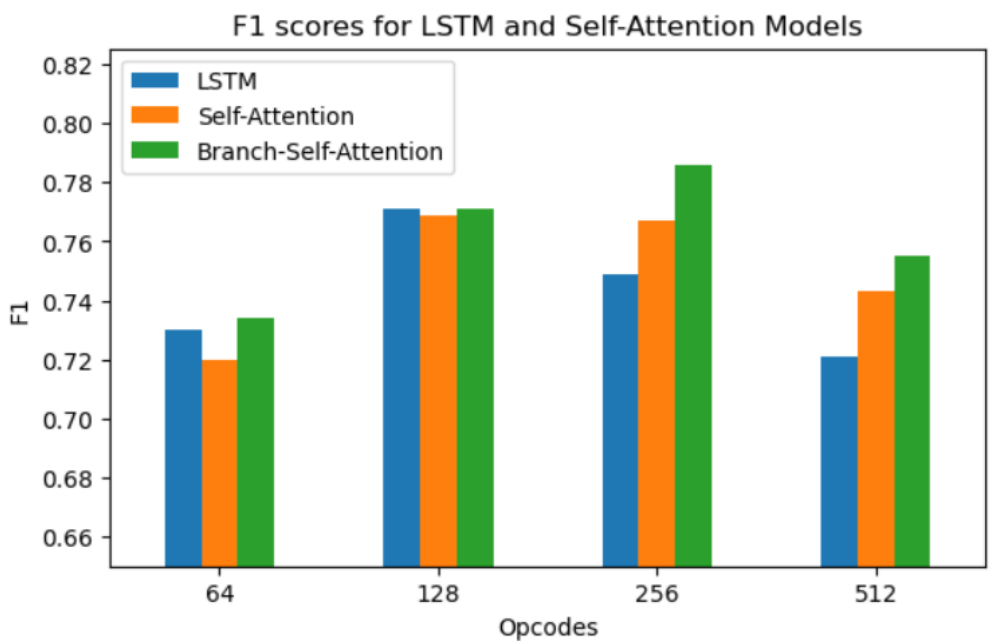
Metrics	Initial vulnerability types					
	cl. 1	cl. 2	cl. 3	cl. 4	cl. 5	cl. 6
Loss	0.08	0.05	0.07	0.09	0.01	0.09
Precision	0.99	0.97	0.99	0.99	1.00	0.98
Recall	0.88	0.96	0.91	0.85	0.99	0.88
F1 score	0.93	0.96	0.95	0.91	0.99	0.93
FPR	0.00	0.01	0.00	0.00	0.00	0.00
FNR	0.12	0.06	0.10	0.15	0.01	0.12

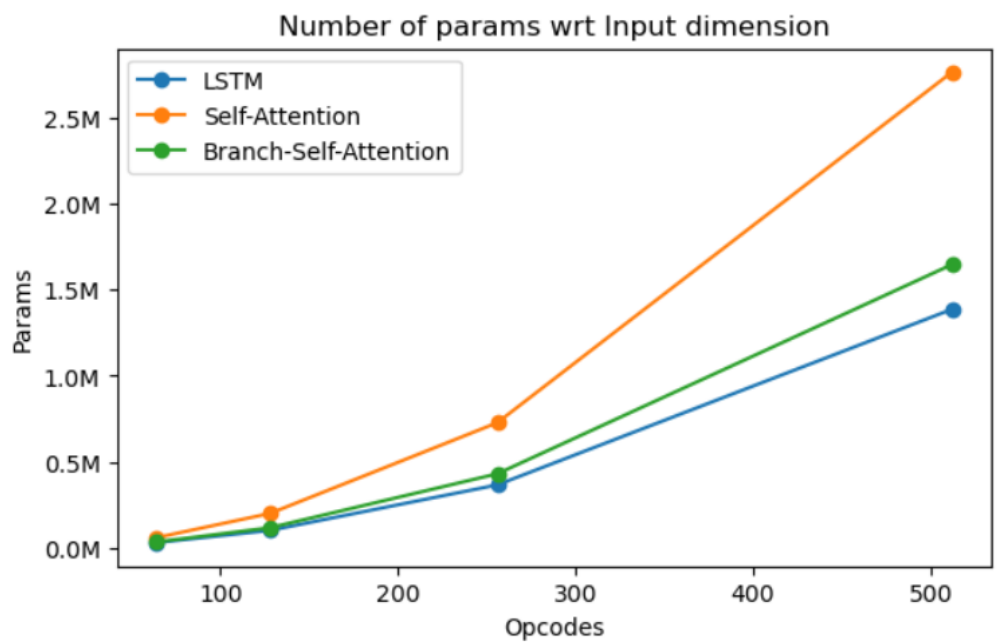
- **High performances;**
- **Complex model (high number of parameters);**

Our project

- We first implemented a **self attention model** and an **ESCORT model** where each branch has a self attention layer;
- Then we compared these results with a **LSTM baseline** for **different input length,** by simply changing the number of the first opcodes taken as input;
- The best model (according to the performances on the test set) was chosen to be **tested with different input windows** (first opcodes, last opcodes and so on) to see the behaviour of the model;
- We also tested the different window wrt to the F1-score of each class;
- Finally we made some **conclusions** about performances of our models and a comparison between our work and the two papers;
- **Future steps**

Our three models

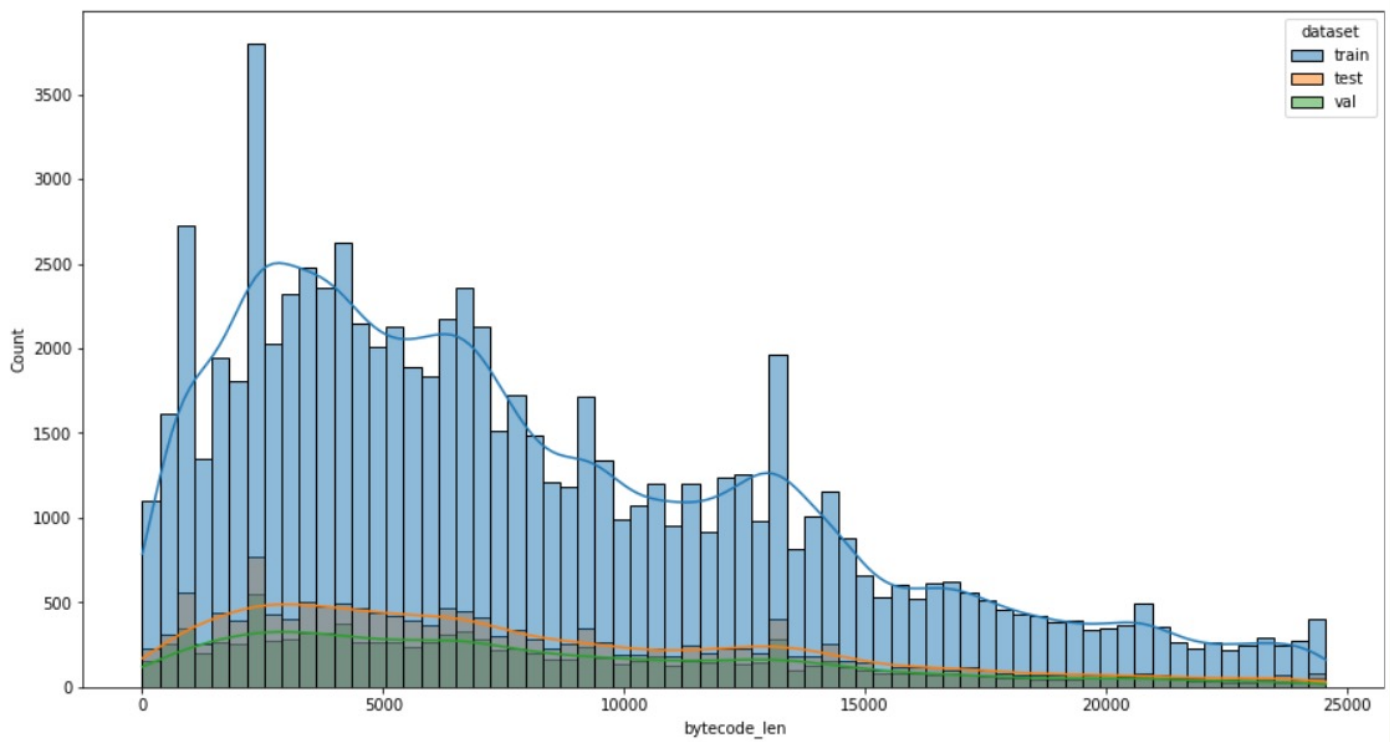


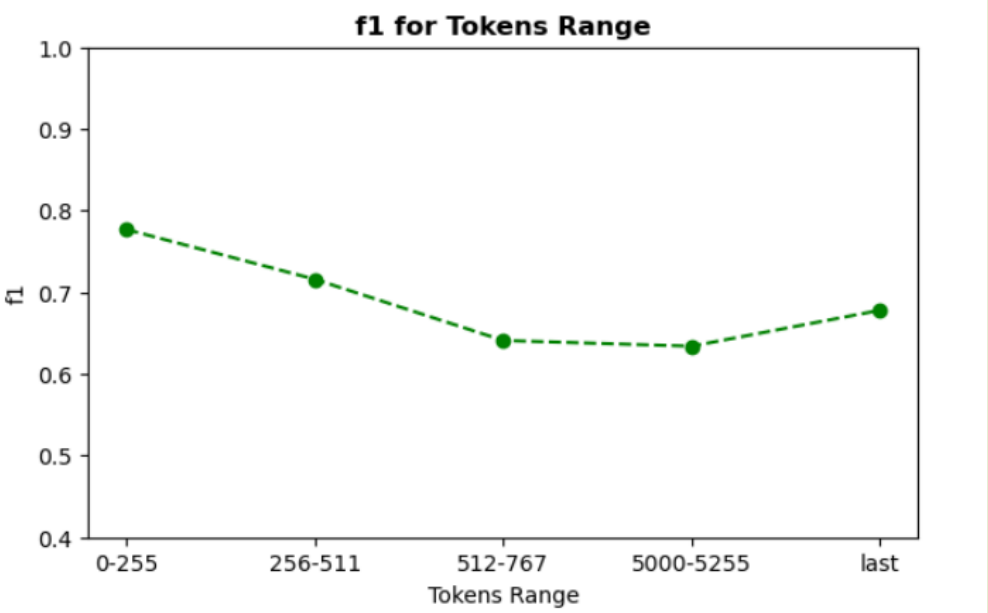


- Best performances achieved by the Branch-Self-Attention (ESCORT + self attention) with an input size of 256 bytecodes;
- The Branch-Self-Attention has always the best performances wrt the input size;
- The LSTM the worst, and it performance decreases as the input size grows (due to its nature);
- By increasing the input size, the Self Attention Model is the one that increases its number of parameters the most, while the other two models have almost the same number of parameters.

Changing the input window

Then, we tried different windows of the same length (256) for the Self-Attention Branch model. In particular, our computational resources didn't allow us to iterate the window over all the input, since a lot of instances have more than 5k of bytecodes:

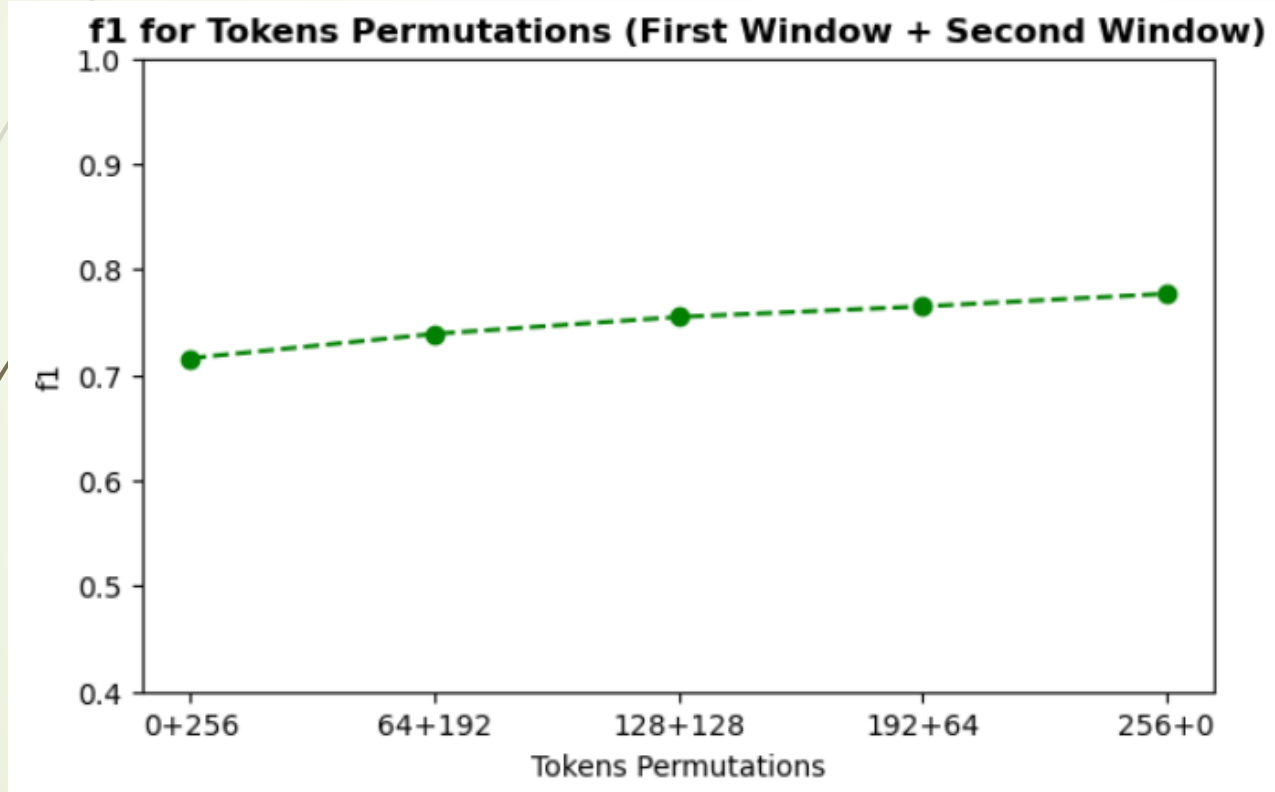




The best F1 score corresponds to the first window (first 256 bytecodes), and the second best performance to the second window.

Different combinations of windows

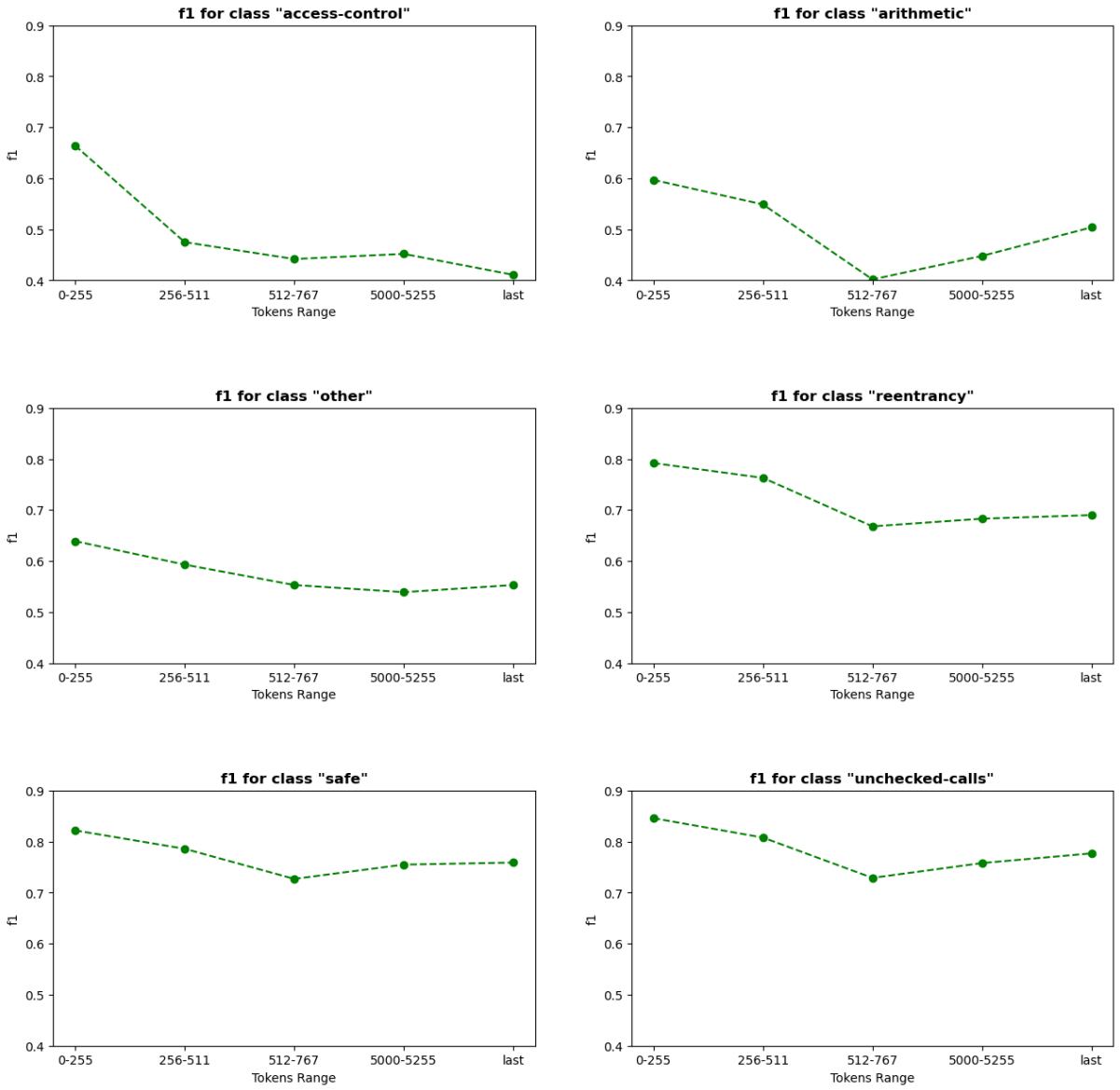
Finally, we combined different portions of the first two windows to see if the performance increases wrt taking the first 256 bytecodes



Again, the best performance corresponds to the first window.

Performance of each class

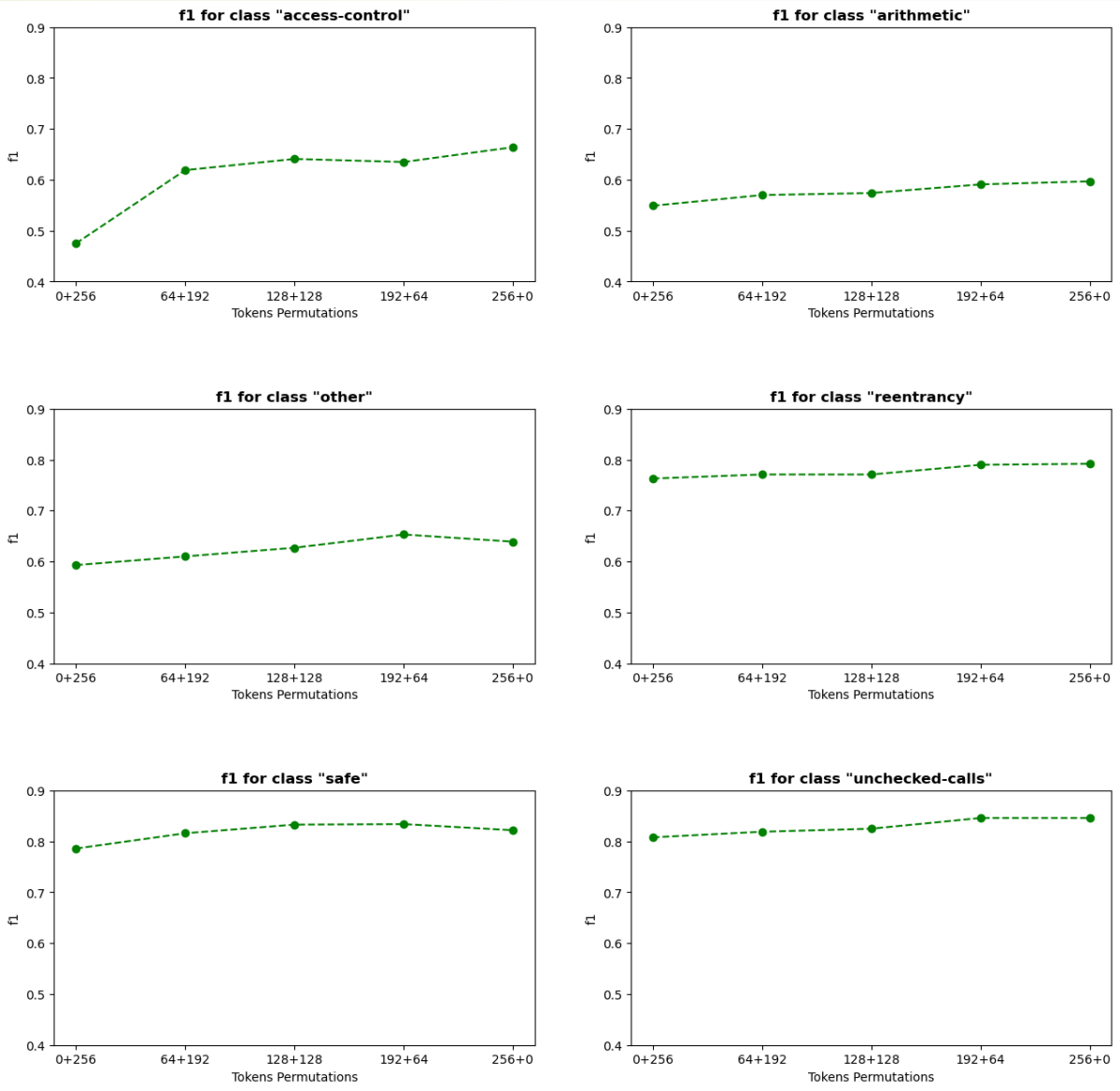
We also analyzed the F1 score of each class wrt the different windows:



As we can see from the results, all the classes show the same trend: by sliding the window, the F1 score decreases, and the best result correspond always to the first window.

Performance of each class

We also tested different combinations of the first two windows for each class:



Again, by combining the first two windows (the best ones) the performance doesn't improve.

Conclusion

- Our work reaches the best performance with the Branch-Self-Attention model by taking as input the first 256 opcodes, which performed with a micro F1 score equal to 0.79;
- This result is in line with that of the first paper, but it is worse than the original ESCORT;
- However, our model has significantly fewer parameters than the other two (430k vs), and we also used a much smaller input size (256 vs 16384 & 4100);
- We also demonstrated that the first window of 256 tokens is always the best wrt to other windows (second, third, middle and last) of different input length (64,128,512), and also that the combinations of the best two windows doesn't improve the performance of the model.

Future Steps

- As future step, it would be interesting to have higher computational resources:
  - To train more complex model;
  - To try larger input size;
  - To iterate the window over all the input