**BTP Report**

**Project Title: Hardware Security using AI/ML Techniques**

**Project Mentor: Dr Binod Kumar**

**By:** [Praneet Thakur (B19CSE066)](mailto:thakur.5@iitj.ac.in)

**Introduction:**

Globalization of hardware design and fabrication processes have raised serious concerns about hardware-based attacks. Hardware has been always assumed to be the guarantee of trustworthiness in cryptographic algorithms and security protocols. However, several backdoors have been reported in the last years, especially in military contexts.

Hardware Trojans (HTs) are malicious changes made to integrated circuits in order to disrupt their functional behaviour. They are made up of two major components: the trigger, which activates the malicious behaviour under certain conditions, and the payload, which performs the malicious tasks.

The triggers can include

(i) functional based conditions, e.g., a specific value or a sequence of values, which activates the payload once

it has been observed on a certain register or port,

(ii) physical-based conditions, e.g., reaching a

value of temperature or power,

(iii) time-based conditions, e.g., a certain number of cycles or

operations that must be counted.

Payloads typically exhibit even more diversity, e.g., leakage of

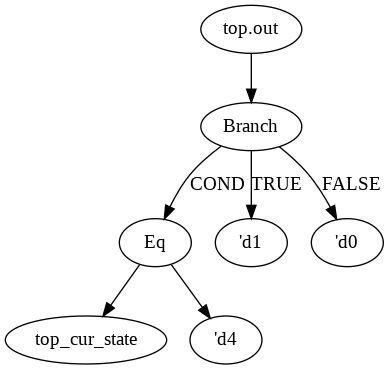
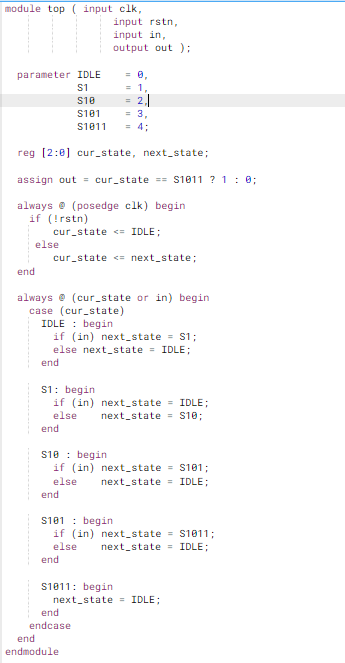
information, data corruptions, performance loss, etc

HTs can be added during every phase of the fabrication process, e.g., design or synthesis, and they are designed to remain silent during the whole verification and testing phase, thus causing the failure of the standard verification approaches. HTs are more and more inserted at RTL because, at this level of abstraction, attackers have high flexibility to implement any malicious function. In this project we have focused on detection of HTs inserted in the RTL phase.

**Introduction:**Model the circuit using graph neural networks(GNN) to detect hardware Trojan.

Verilog Code⇒ Data Flow Graph(DFG) and Abstract Systax Tree(AST) => GNN =>Hardware Trojan detection

DFG example: Code and DFG



**Steps:**

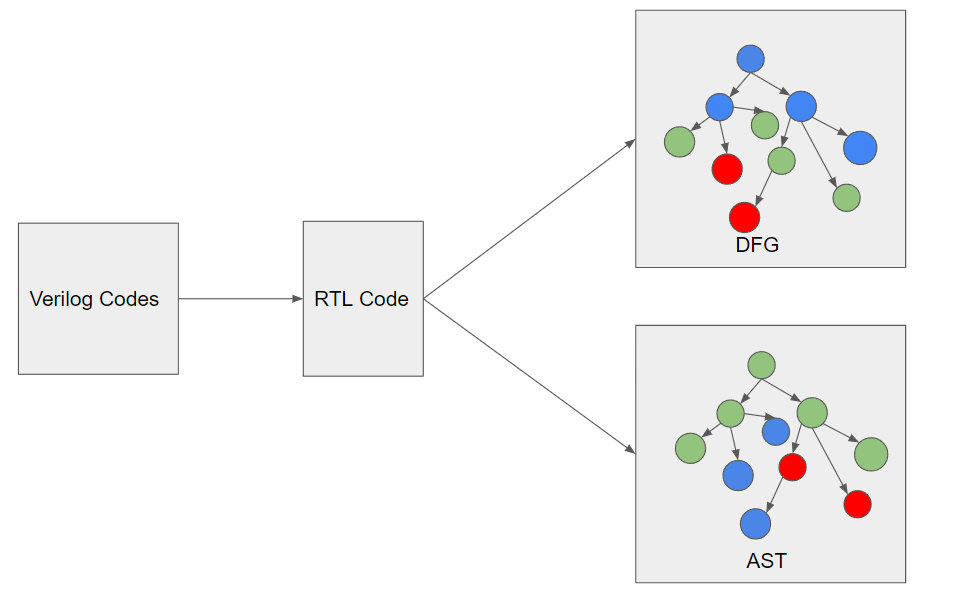
i) Extracting DFG and AST from the hardware designs.

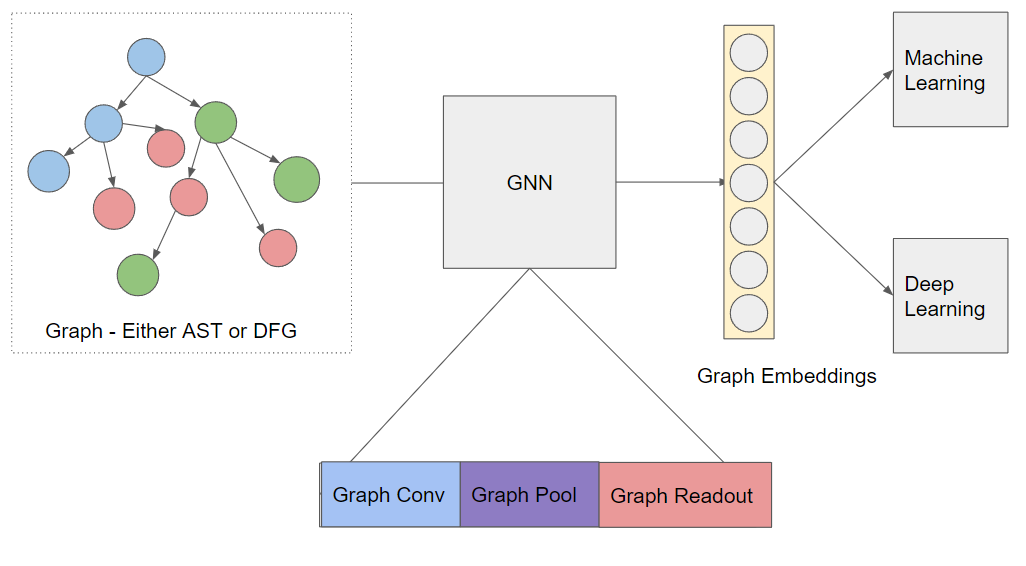
ii) The DFGs and ASTs passed to GNN to generate graph embeddings.

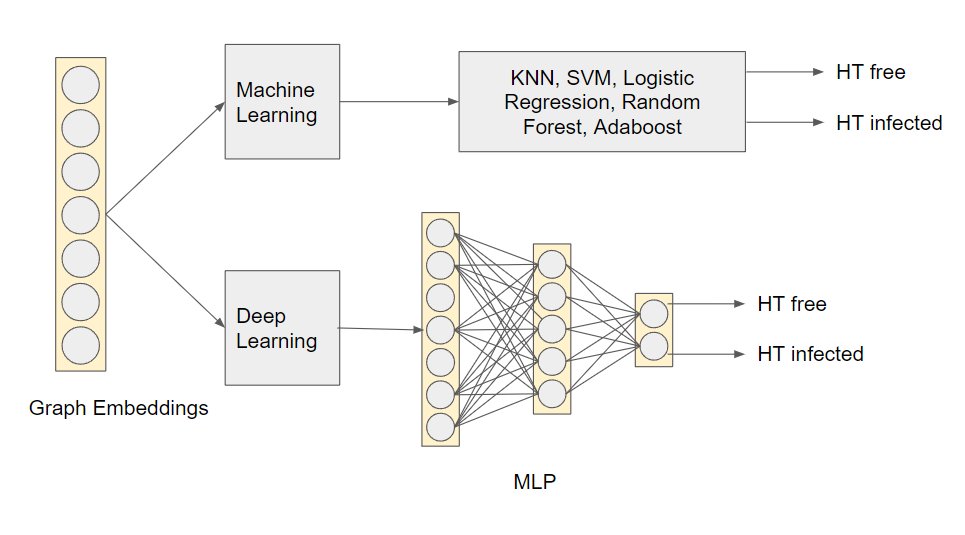
iii) Machine learning models on the generated graph embeddings to detect wheter trojan infected or not.(AST, DFG)

iv) MLP trained on the embeddings to detect whether trojan infected or not.(DFG, AST, AST+DFG)

**Workflow Diagram:**





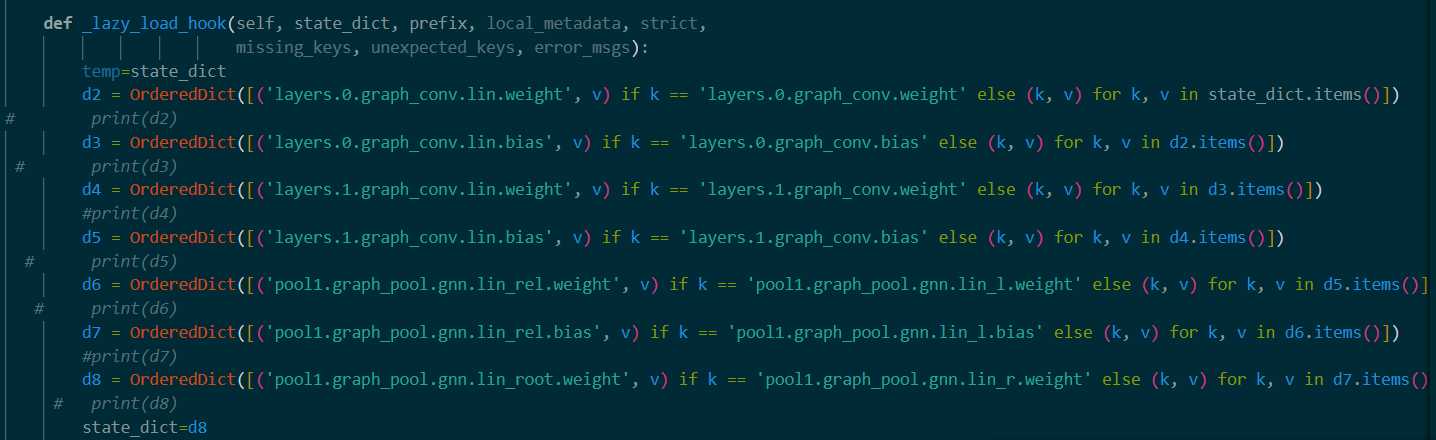


**Difficulties faced:**

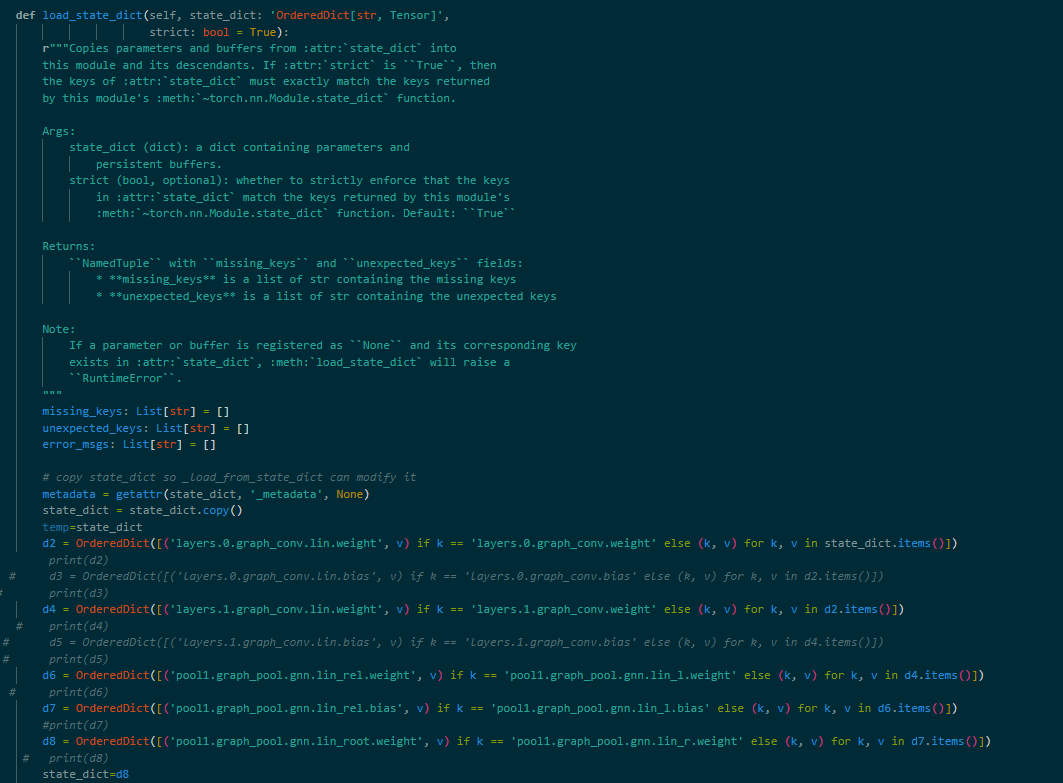
There were some difficulties faced. Tried changing environments multiple times but nothing worked. So, I had to patch some files in pytorch and pytorch-geomertic in order to get embeddings.

The patched portions are:

i)usr/local/lib/torch-geometric/nn/Dense/linear.py



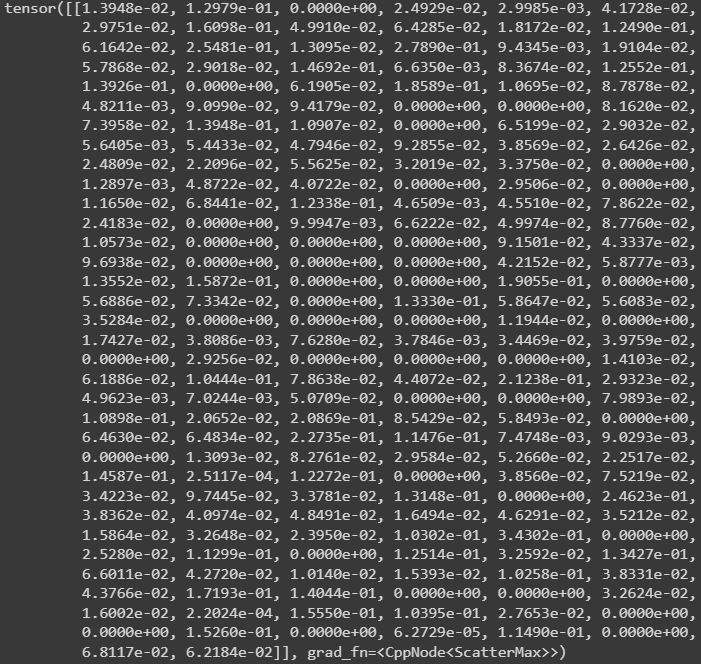
ii)usr/local/lib/torch/nn/Module/module.py

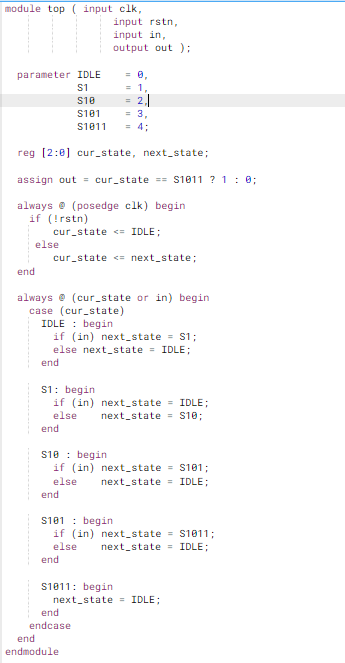




**Example Embedding:**

Hardware: det\_1011





**Graph saint:**

* GNN requires [GraphSAINT](https://github.com/GraphSAINT/GraphSAINT) to perform node classification . I have used the TensorFlow implementation of GraphSAINT.
* Since the size of dataset is small I have used the graph saint subgraph matching technique to increase the training for the model.
* In GraphSAINT, subgraphs are sampled from the original graph, and a full GNN is constructed for each subgraph.

**Results:**

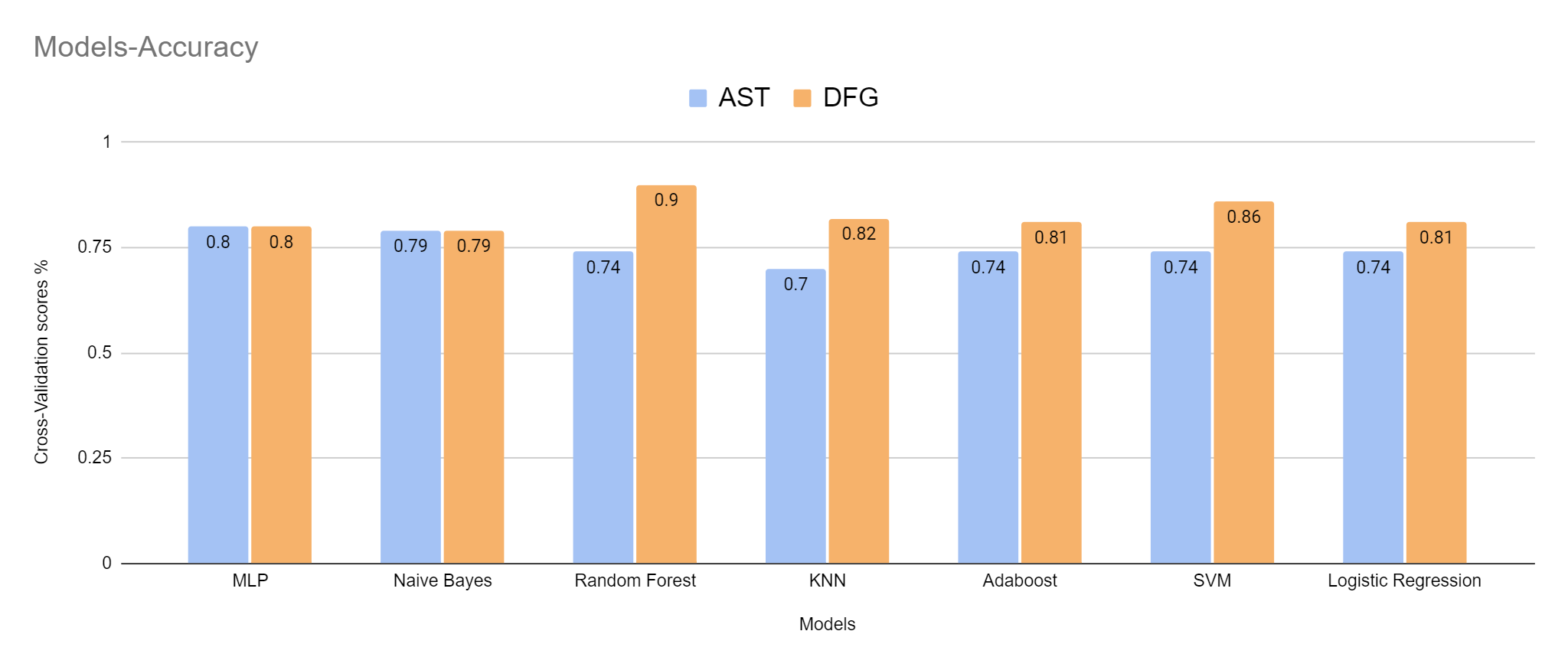
I have tried to optimize the gnn model by trying out different set of hyperparameters. Two such set of parameters are shown ahead with different sets of results. The metrics for evaluating the model are accuracy, precision score, recall and f1 score.

GNN configuration:

| **Architecture 1** | **Training** |
| --- | --- |
| Total number of layers - 2  Pooling type: “topk” graph pooling  Readout\_type - “max”  Activation - Relu | Optimiser - Adam  Dropout - 0.5  Learning rate - 0.001  Batch\_size - 4 #number of graphs in a batch  Epochs - 200 |

Results for GNN1:

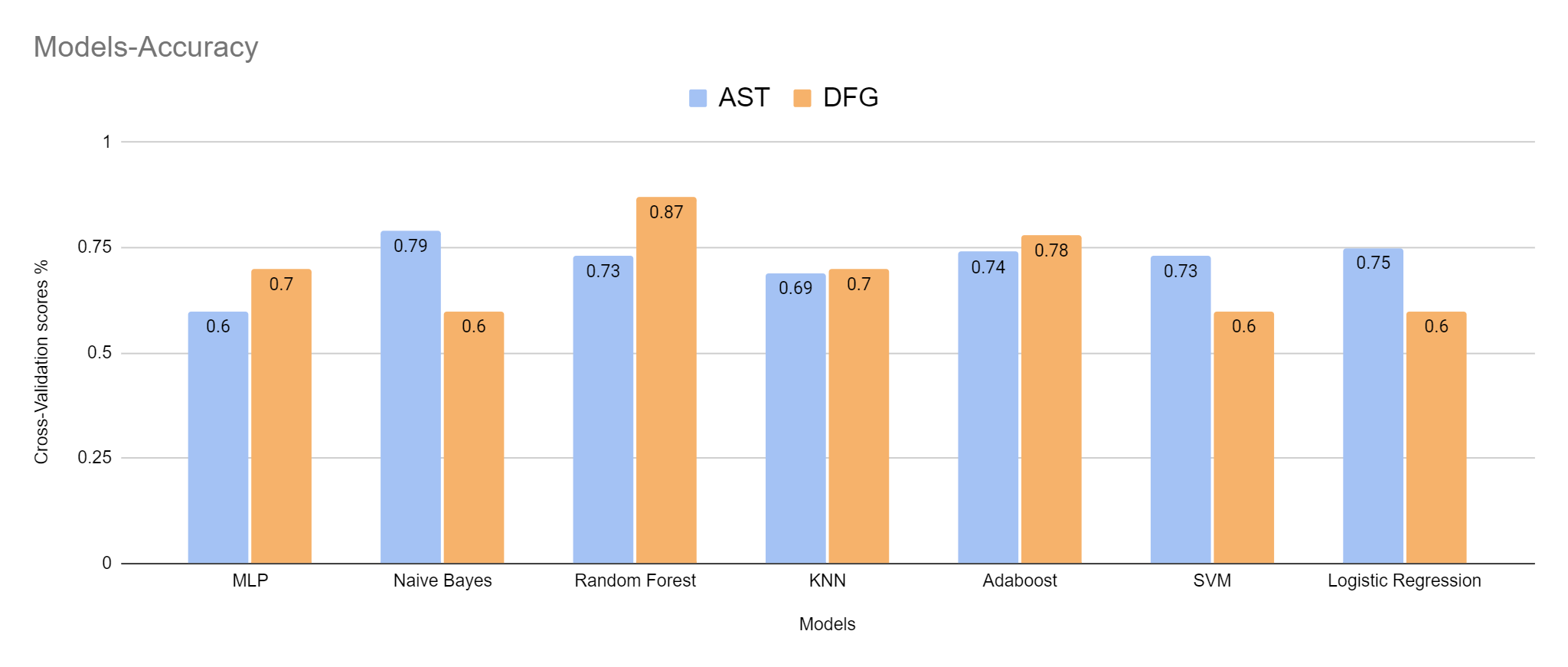
| **Model** | **AST Accuracy ,Precision, Recall, F1score** | **DFG Accuracy,**  **Precision, Recall, F1score** |
| --- | --- | --- |
| MLP | accuracy:0.80, precision: 0.7500, recall: 1.0000, f1 score: 0.8000 | 0.80 precision: 0.7500, recall: 1.0000, f1 score: 0.8000 |
| KNN | 0.70, 0.6, 1.0, 0.749 | 0.82, 0.6, 1.0, 0.749 |
| SVM | 0.74, 0.75, 1.0, 0.857 | 0.86, 0.75, 1.0, 0.857 |
| Random Forest | 0.74, 0.6, 1.0, 0.749 | 0.90, 0.75, 1.0, 0.857 |
| Adaboost | 0.74, 0.75, 1.0, 0.857 | 0.81, 0.75, 1.0, 0.857 |
| Logistic Regression | 0.74, 1.0, 1.0, 1.0 | 0.81, 1.0, 1.0, 1.0 |
| Naive Bayes | 0.79, 1.0, 1.0, 1.0 | 0.79, 0.75, 1.0, 0.857 |



| **Architecture 2** | **Training** |
| --- | --- |
| Total number of layers - 3  Pooling type: “topk” graph pooling  Readout\_type - “max”  Activation - Relu | Optimiser - Adam  Dropout - 0.4  Learning rate - 0.001  Batch\_size - 4  Epochs - 200 |

Results for GNN2-

| **Model** | **AST Accuracy ,Precision, Recall, F1score** | **DFG Accuracy, Precision, Recall, F1score** |
| --- | --- | --- |
| MLP | Accuracy: 0.60, precision: 0.6000,recall: 1.0000,f1 score: 0.7500 | 0.70 precision: 0.6000,recall: 1.0000,  f1 score: 0.7500 |
| KNN | 0.69, 0.6, 1.0, 0.749 | 0.70, 0.75, 1.0, 0.857 |
| SVM | 0.73, 0.75, 1.0, 0.857 | 0.60, 0.6, 1.0, 0.749 |
| Random Forest | 0.73, 0.75, 1.0, 0.857 | 0.87, 0.75, 1.0, 0.857 |
| Adaboost | 0.74, 0.75, 1.0, 0.857 | 0.78, 0.75, 1.0, 0.857 |
| Logistic Regression | 0.75, 1.0, 1.0, 1.0 | 0.60, 0.6, 1.0, 0.749 |
| Naive Bayes | 0.79, 0.75, 1.0, 0.857 | 0.60, 0.6, 1.0, 0.749 |



**Conclusion:**

Through this project I have put forward an approach that can be used for the detection of HTs using DFG, GNN and ML/DL models.

**Future Works:**

One should use the same approach on a bigger dataset. We can also play around with GNN structure to get a more efficient model. One can use ensemble learning and one can use few shot learning using siamese network. One should also consider using CFG for the task.

**References:**

[**https://pypi.org/project/pyverilog/**](https://pypi.org/project/pyverilog/)