Master Thesis

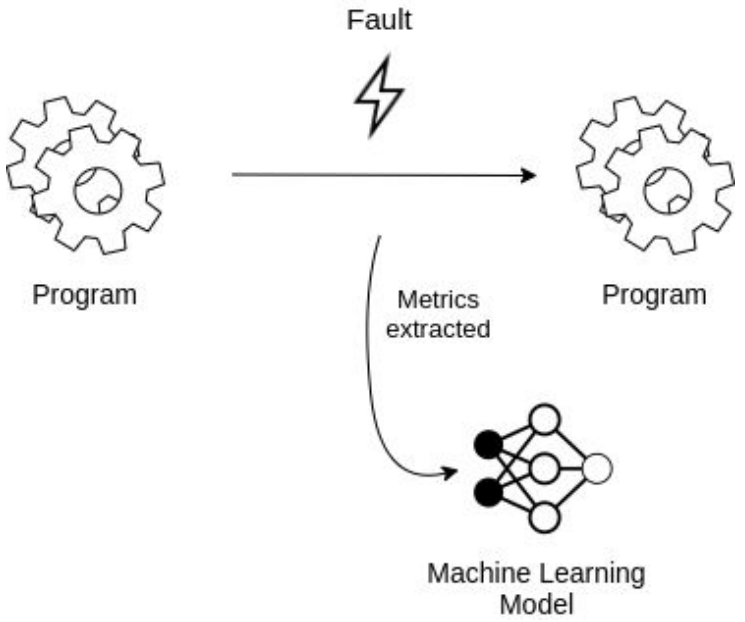
Virtual Tool-boxing for Robust Management of Cross-layer Heterogeneity in Complex Cyber-physical Systems

Candidate: Simone Dutto

Supervisors: prof. Stefano Di Carlo, prof. Alessandro Savino

Abstract

The final goal of this work is to analyze what can be done in terms of fault detection using hardware metrics and machine learning.



AGENDA

01
Simulation Environment
How I obtained the data

02
Data analysis and Models
How I used data to create machine learning models

03
OS Implementation
How I exploited the models to create a fault detection system

04
Conclusion and Future Works
Final thoughts and possible extensions

01

SIMULATION ENVIRONMENT

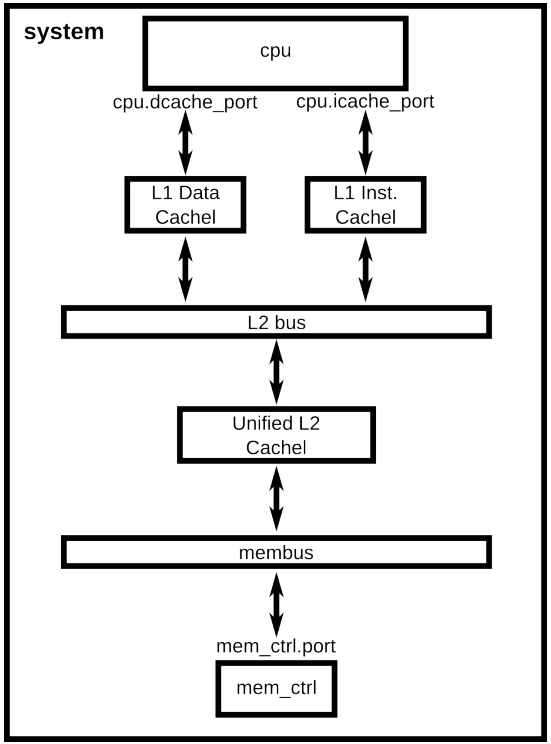
gem5

gem5 is an hardware simulator able to:

- Simulate hardware architecture
- Extract metrics from components
- Customizable to inject faults
- Run Linux Distribution

Simulated System

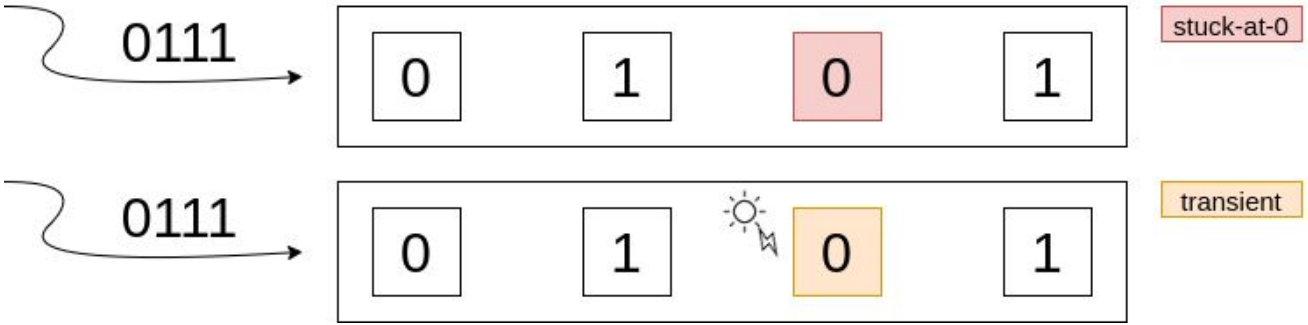
- Atomic Simple CPU
- x86 architecture
- L1-L2 cache
- Full System Simulation
- MiBench programs



FIMSIM

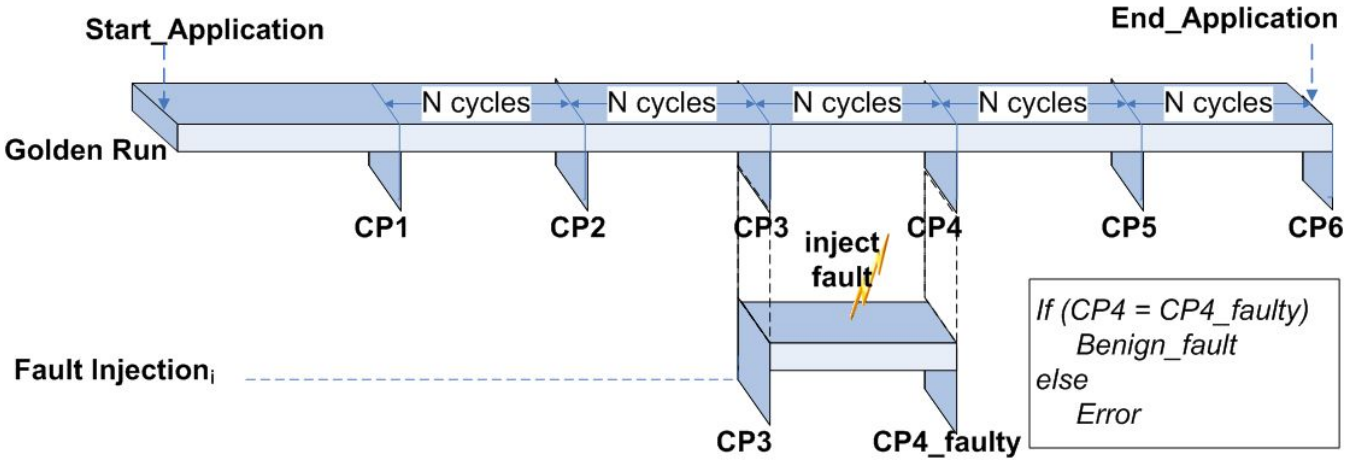
FIMSIM is a gem5 extension able to simulate the occurrence of different types of faults:

1. Stuck-at-0
2. Stuck-at-1
3. Transient



Fault Injection Results

The fault injection is evaluated comparing golden runs with faulty runs.



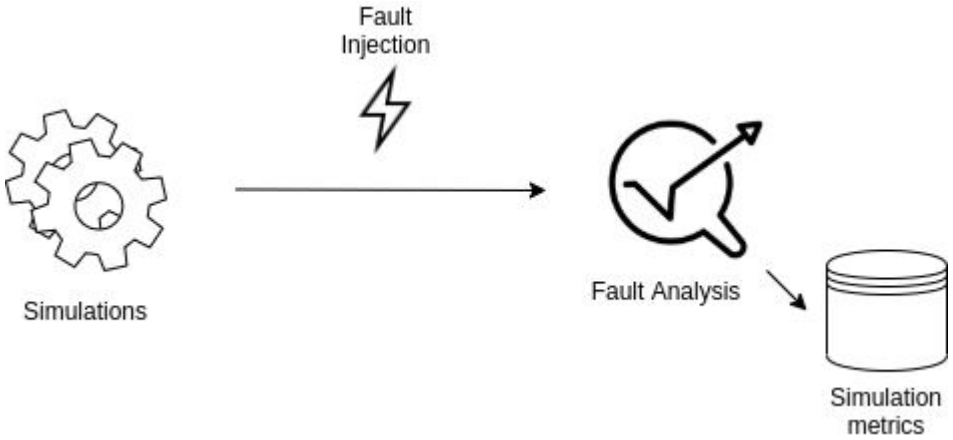
Data collected

- Stuck-at-0 (10k runs, 70% crash rate)
  - bitcount
  - qsort
  - basicmath
  - ssearch
- Stuck-at-1 (10k runs, 80% crash rate)
  - bitcount
  - qsort
- Transient (10k runs, 1% crash rate)
  - bitcount

02

Data Analysis and Model

Data




```
---------- Begin Simulation Statistics ----------
sim_seconds  0.047551  # Number of seconds simulated
sim_ticks  47550712000  # Number of ticks simulated
```

Data Preparation

Repeat for each binary and each fault

PHASE 1

Statistics files are parsed to obtain different dictionaries, which are then merged

PHASE 2

Drop columns with >20 NaN and entries with all NaN

PHASE 3

Fill NaN with column mean

PHASE 4

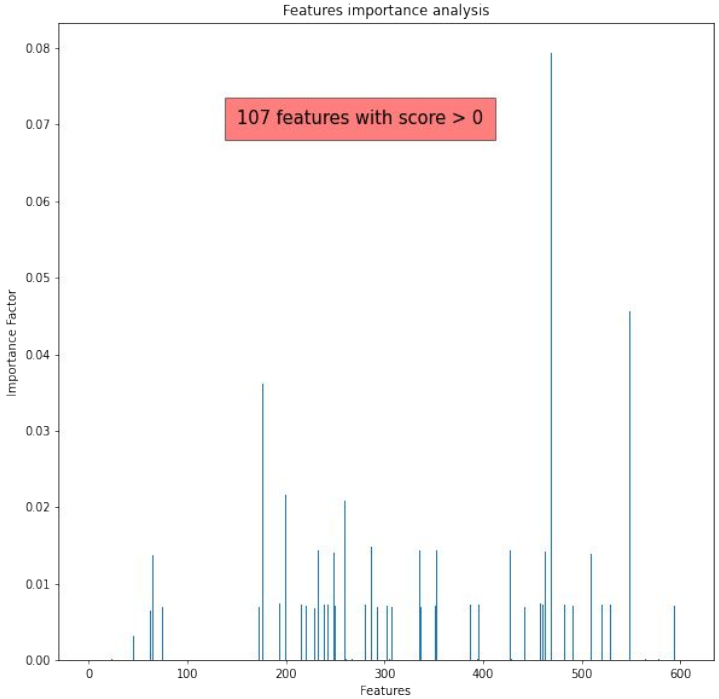
Remove all columns with <0.1 variance

Feature Selection

Feature importance score using Random Forest Classifier.

- Avoid searching just for linear relationships
- Keep the meaning of features

I have selected the top 19 corresponding feature importance scores.



Binaries and domains

Different binaries carry different metrics distribution.

So different models and techniques are tried to cope with the differences.

Basicmath: Simple math lib operations	**QSort:** Quick sort on an array	**Search**: Search a string inside a string	**Bitcount:** Count number of 1-bit in a number

Model-FFNN

Simple Neural Network with 19-features selection trained on a domain for 50 epochs.

Exploit transfer learning (10 epochs) to adapt on different domains.

Very good convergence, fast to train and evaluate.

Binary	Precision: baseline	Precision: after	Recall: baseline	Recall: after
basicmath	1.00	/	0.74	/
qsort	0.38	1.00 (+0.62)	1.00	0.82(-0.18)
search	0.46	0.82 (+0.36)	1.00	0.33(-0.67)
bitcount	1.00	1.00(+0.0)	0.73	0.74(+0.01)

Model-SSAE

Sparse Stacked AutoEncoder, based on CNN and full features selection.

There the auto-encoder is trained to with unlabelled data on non-faulty runs.

The principal concept is to have the encoder able to distantiate faulty and non-faulty runs the best it can before training the classifier on actual labelled data.

Binary	Precision: before	Precision: after	Recall: before	Recall: after
basicmath	1.00	\	0.80	\
qsort	1.00	1.00(+0.0)	0.20	0.78
search	0.86	0.83(-0.03)	0.11	0.32
bitcount	0.12	1.00(+0.88)	0.23	0.77

Model-DANN

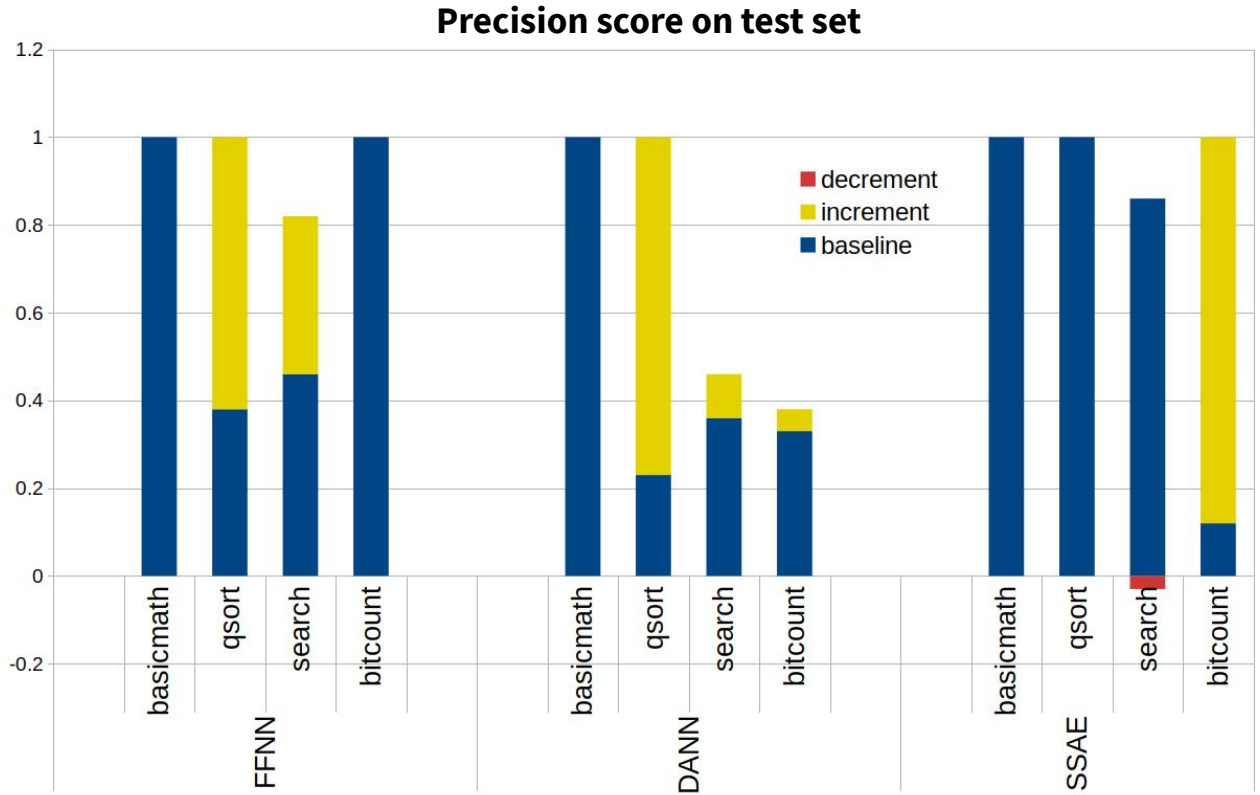
Domain Adversarial Neural Network, based on CNN and full features selection.

The idea is to have source and target domains, the first entries are labelled, the second are not.

The catch here is to train the feature extractor to perform on the task without considering the domain using an adversarial approach.

Binary	Precision: before	Precision: after	Recall: before	Recall: after
basicmath	1	\	0.80	\
qsort	0.23	1(+0.77)	1	0.77
search	0.36	0.46(+0.10)	0.8	1
bitcount	0.33	0.38(+0.05)	0.8	1

Results


Precision score on test set
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
decrement
increment
baseline
basicmath
qsort
search
bitcount
FFNN
basicmath
qsort
search
bitcount
DANN
basicmath
qsort
search
bitcount
SSAE

03

OS Implementation

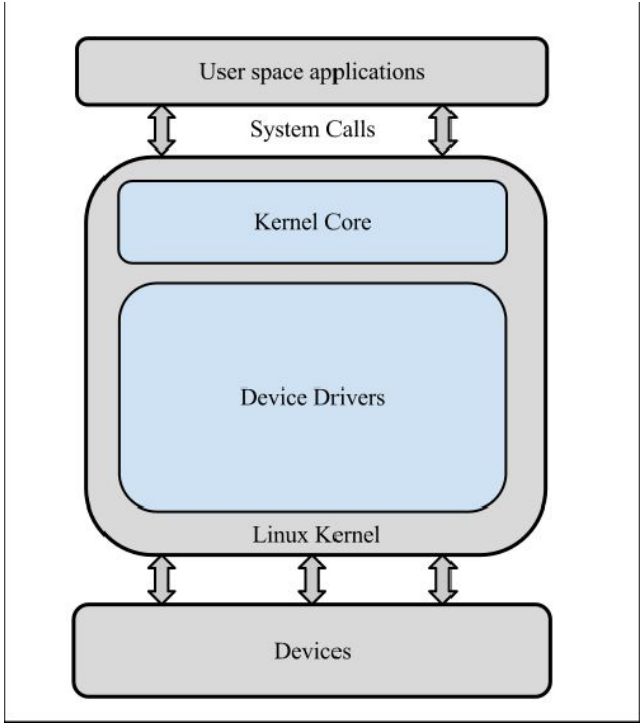
Kernel Module

Fast as the functions were implemented in the kernel.

The change is not bounded to the kernel.

Loaded only if needed.

Easy interactions with user-level applications.



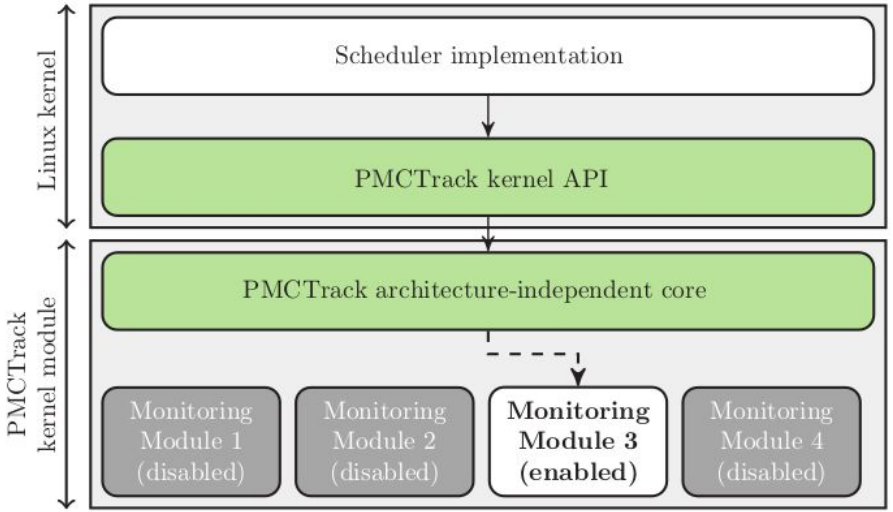
PMCTrack

Kernel module to extract Performance Monitor Counters.

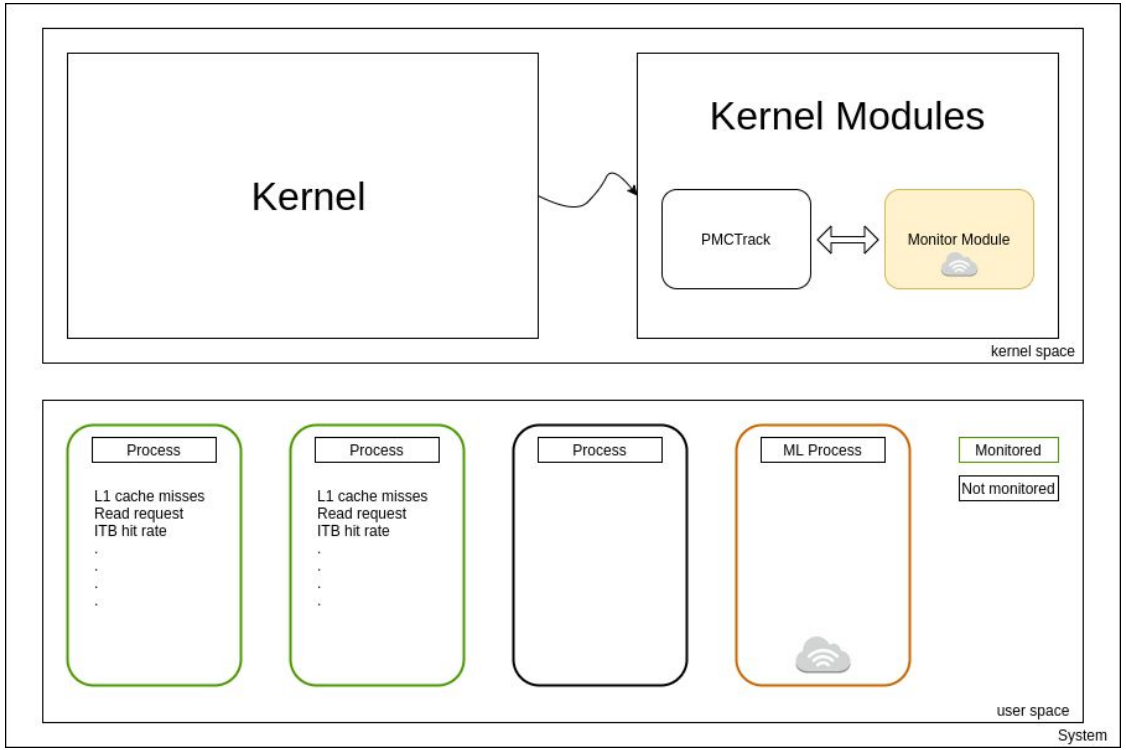
Minor changes to the kernel, all implemented at kernel module level.

Expose metrics to kernel and user-level applications.

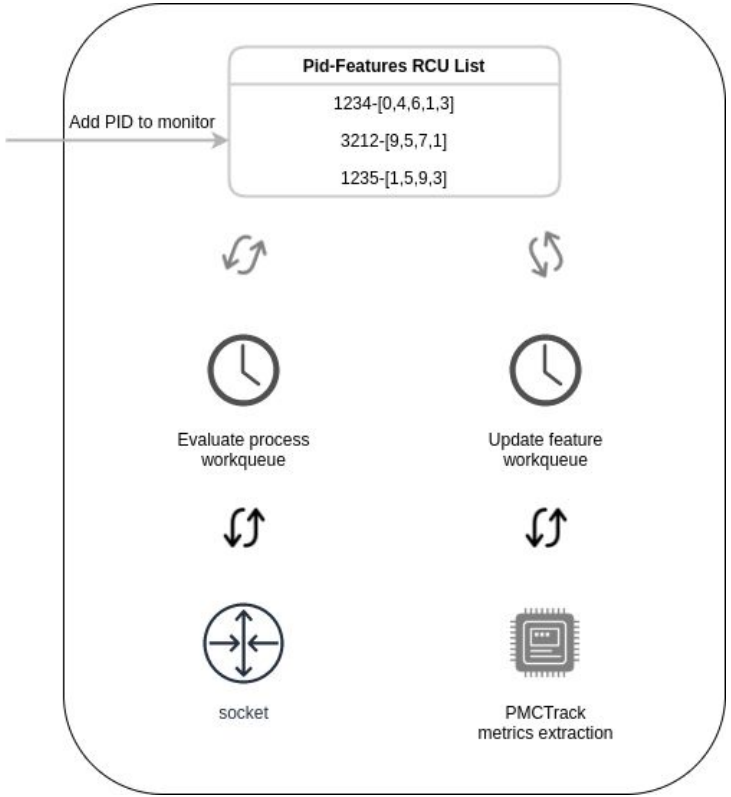
Process-wise.



Monitor Framework



Monitor Module



04

Conclusion and Future works

Conclusion

It is definitely possible to state the fault detection is doable thought hardware metrics.

Even a simple model works very well with a 19-features selection.

The OS implementation needs other metrics implemented to be working.

In addition, a lot can be done to improve the faults detection framework.

Future works

Collect more data with different binaries and fault types.

Modify simulation output to resemble stream output.

Build time-based models to work with stream data.

Add metrics support to PMCTrack.

Incapsulate the model evaluation kernel-side.

Performance analysis with monitor framework.

The work on machine learning models has been submitted to a conference.

Contacts

To have more details on the whole project, it is possible to:

- Consult my thesis
- Have a look at my GitHub page with all the code related at *github.com/SimoneDutto*
- Write me an email *simonedutto8@gmail.com*