# Updates on Causality Analysis and Noise Detection in Log sequences and Software security vs Observability

Fateme Faraji Daneshgar

Polytechnique Montréal

DORSAL Laboratory

# Agenda

- Updates on llm-augmented causality analysis in log sequences
- Updates on noise log elimination
- Updates on software security mechanisms vs observability

# Causality Analysis

identify **cause-and-effect relationships** between events, rather than simple correlations.

- **Correlation** answers: *“Which events tend to occur together?”*
- **Causality** answers: *“Which events directly influence or trigger others?”*

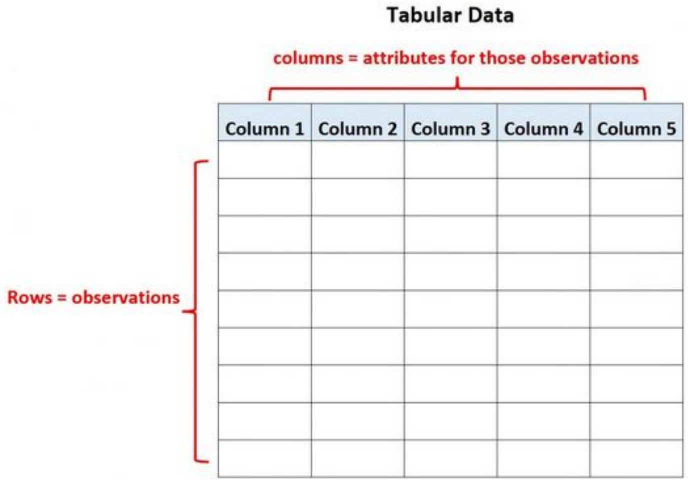
In the context of system logs:

- **Root cause identification**
- **Reduction of redundant alerts**
- **Better debugging and incident response**
- **Explainable models** for system behavior (who causes what, and why)

# Traditional Causality Analysis approaches

## 1. Constraint-Based Methods

- PC algorithm, conditional independence tests
- Data is in **tabular form**
- **Limitation for Log Sequences**
  - Loss of temporal information
  - High number of columns (log templates)
  - Endless computations



# Traditional Causality Analysis approaches

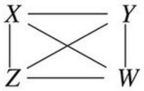
## 2. Time-Series-Based Methods

- Granger causality, temporal statistical models
- **Data :** Dense, regularly sampled time series
- **Limitation for Log Sequences: Sparsity**
  - Log events are often:
    - Irregular, Rare, Bursty
    - Many event pairs have **very few co-occurrences**

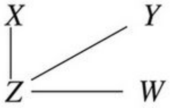
# llm-augmented Causality Extraction

**PC:**

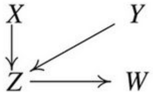
- Having 4 variables {X,Y,Z,W}

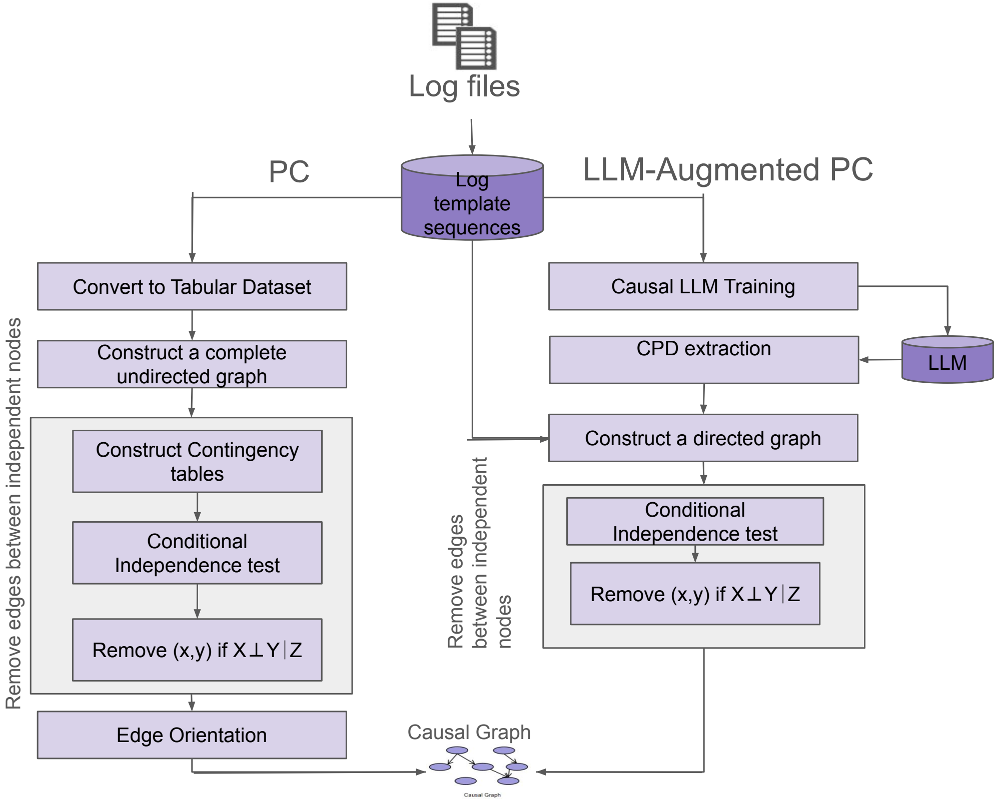



- If X⊥Y∣Z, remove the edge between X and Y




- Edge orientation using d-separation rules

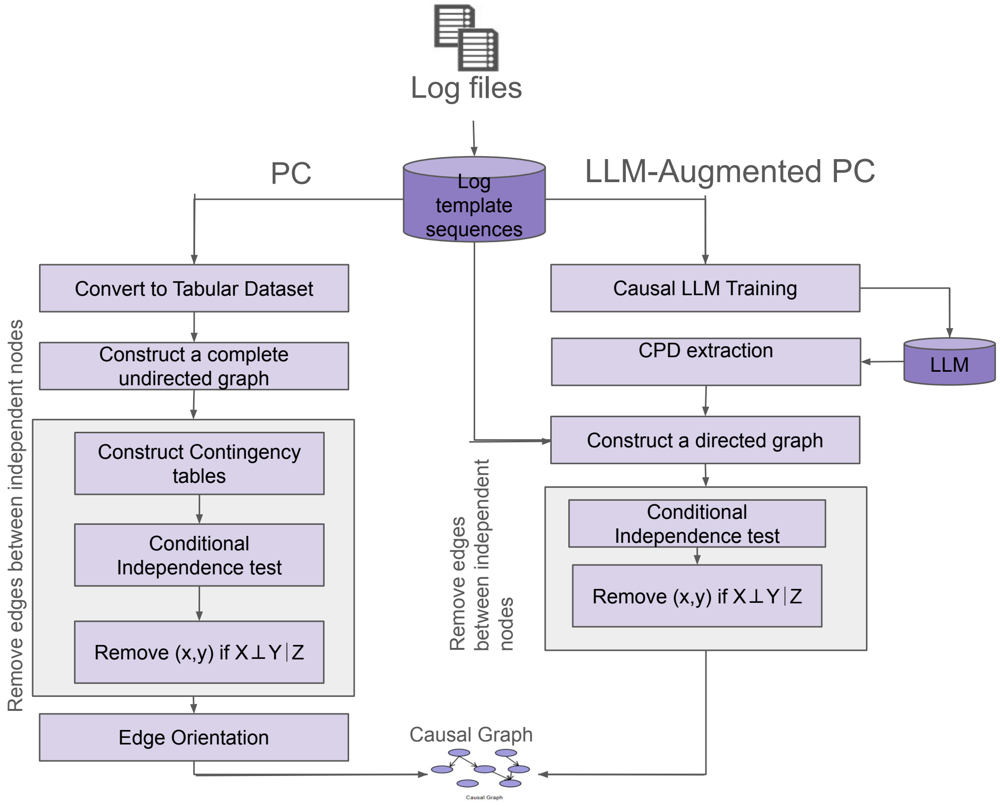


# llm-augmented Causality Extraction

**Main Idea:**

- Train a causal LLM directly on log sequences
- Extract conditional probability distributions (CPDs) from logits
- Eliminate data conversion



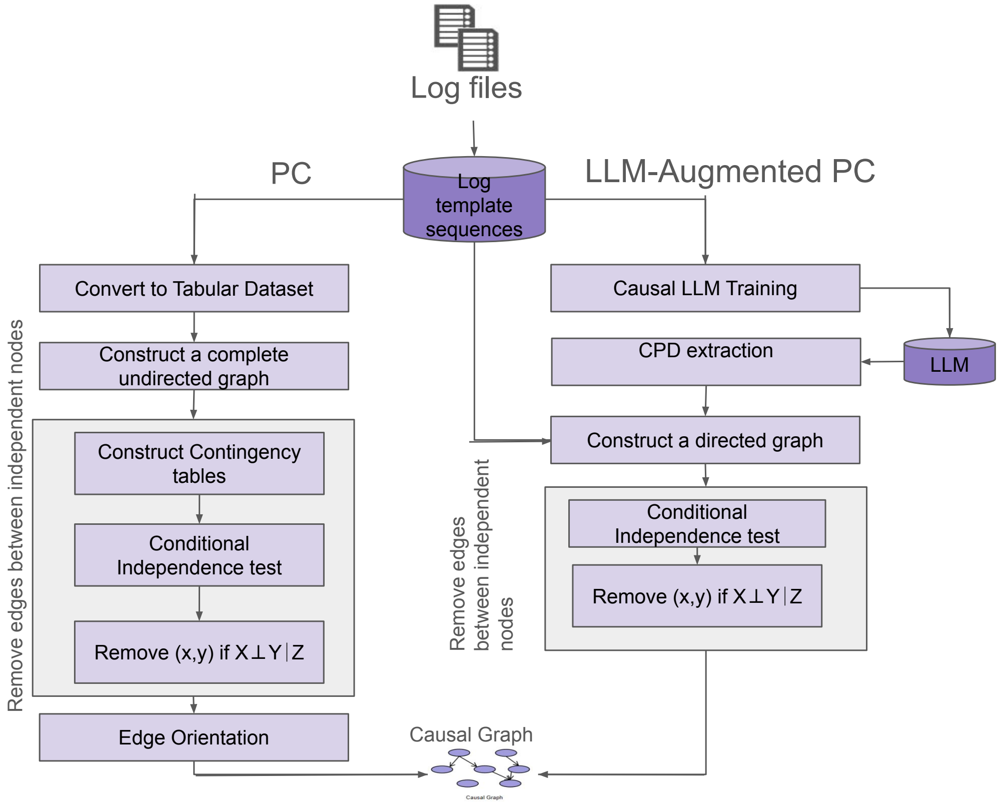
# llm-augmented Causality Extraction

### Llm-augmented PC

- A directed edge between X,Y if:

$$p(X \mid Y) \geq \tau$$

- Eliminating non-causal edge using conditional Independence Test



# Train a causal LLM on log sequences

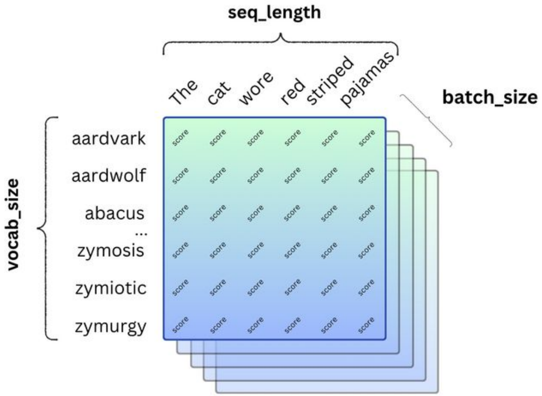
- A **Causal LLM (decoder only transformer)**
  - predict the **next token** in a sequence given all **previous tokens**
  - P(xt|x1 x2 ...xt-1)
  - The model outputs **logits** for each possible token
  - Logits are converted into **probabilities** via **softmax**
- **Training data**
  - Set of **log files**, each containing an order of **log lines**
  - Each log line is a parsed log including **log template**
  - We considered log templates
  - Training data is a sequence set of log templates
  - **Seq: T1, T2, ….Tn**

# Extracting CPDs from LLM

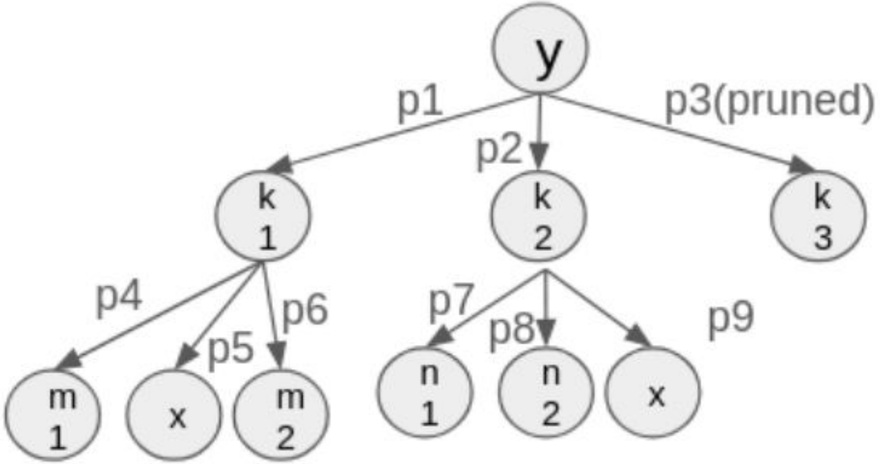
- To compute P(x|y):

$$P(x \mid y) = \sum_{\pi \in \mathcal{P}_{y \to x}} P(\pi)$$

$$P(\pi) = \prod_{i=1}^{t} P(x_{i+1} \mid x_1, \ldots, x_i).$$




logits shape




p(x|y) = (p1*p5)+(p2*p9)

# Conditional Independence test

- Conditional Mutual Information test:

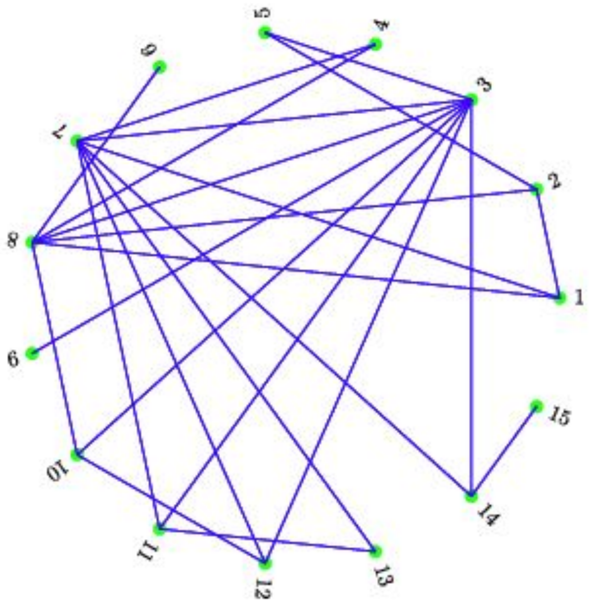
$$I(X;Y \mid Z) = \sum_{x,y,z} p(x,y,z) \log \frac{p(y \mid x,z)}{p(y \mid z)}.$$

$$I(X;Y \mid Z) = \sum_{x,y,z} p(z)\, p(x \mid z)\, p(y \mid x,z) \log \frac{p(y \mid x,z)}{p(y \mid z)}.$$

- Z all subsets of the set of variables occurring before both X and Y
- High value indicate knowing Y increase the probability of X even knowing Z

# The role of Noise logs!

- Causal link + some links between highly connected noise
- Highly connected nodes—–>Nose logs
- We eliminated highly connected nodes



# Experimental Result

- Training Data:
  - 200,000 sequences, average length of 15
  - 1,487 unique template
  - 9 known causal links
- Scalability (the number of conditional independence tests)
  - PC:
    - Number of pairs (x,y) $\binom{n}{2} = \binom{1487}{2} = 1{,}104{,}841$
    - For each pair $2^{n-2}$
    - $1.18 \times 10^{453}$
  - Our approach: 797 conditional independence tests

# Experimental Result

- Detecting causal links (of 9 )
  - PC : 1
  - Granger: 0
  - Our approach:  8

| Method | Precision | Recall | F-1 Score |
|---|---|---|---|
| PC | 0.01 | 0.11 | 0.2 |
| Granger | 0 | 0 | 0 |
| LLM-Augmented PC | **38.4** | **89** | **53.3** |

## SANER 2026

# Extracting Causal Relations from Log Sequences Using Causal Language Models

Vithor Bertalan
Computer and Software Eng. Dept.
*Polytechnique Montréal*
Montréal, Canada
vithor.bertalan@polymtl.ca

Fateme Faraji Daneshgar
Computer and Software Eng. Dept.
*Polytechnique Montréal*
Montréal, Canada
fateme.faraji-daneshgar@polymtl.ca

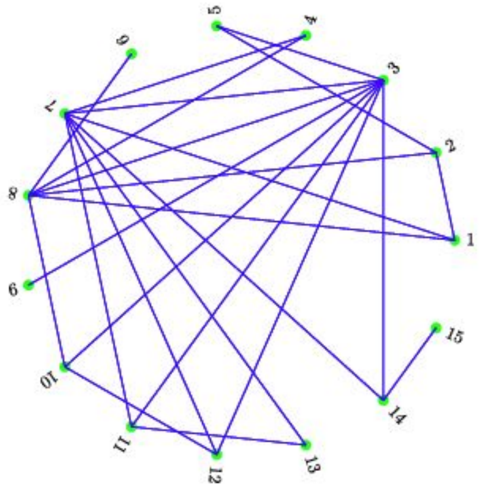
Daniel Aloise
Computer and Software Eng. Dept.
*Polytechnique Montréal*
Montréal, Canada
daniel.aloise@polymtl.ca

***Abstract*—Understanding causal relationships in system logs is crucial for diagnosing complex software behaviors. Traditional causality extraction methods fall into two categories: constraint-based and time-series-based. Constraint-based approaches, such**

line could not have happened without the first. Therefore, we can conclude that there is a causal link between the two. It is important to highlight that the first messages might have

# Updates on Noise log elimination

- Noise logs participate in **many** different error scenarios
- Clustering log sequences to obtain error scenarios
  - Each log file is a sequence of template ids
  - Similarity measure:
    - Number of shared templates/ number of all template in both lines
    - [1 3 6 4 8]  and [3 5 8 9 1] ---- > 3/7

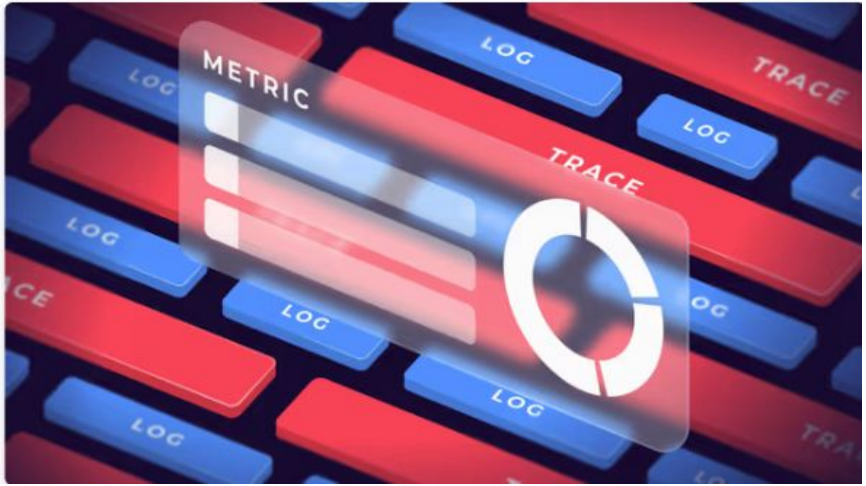


# Updates on Noise log elimination

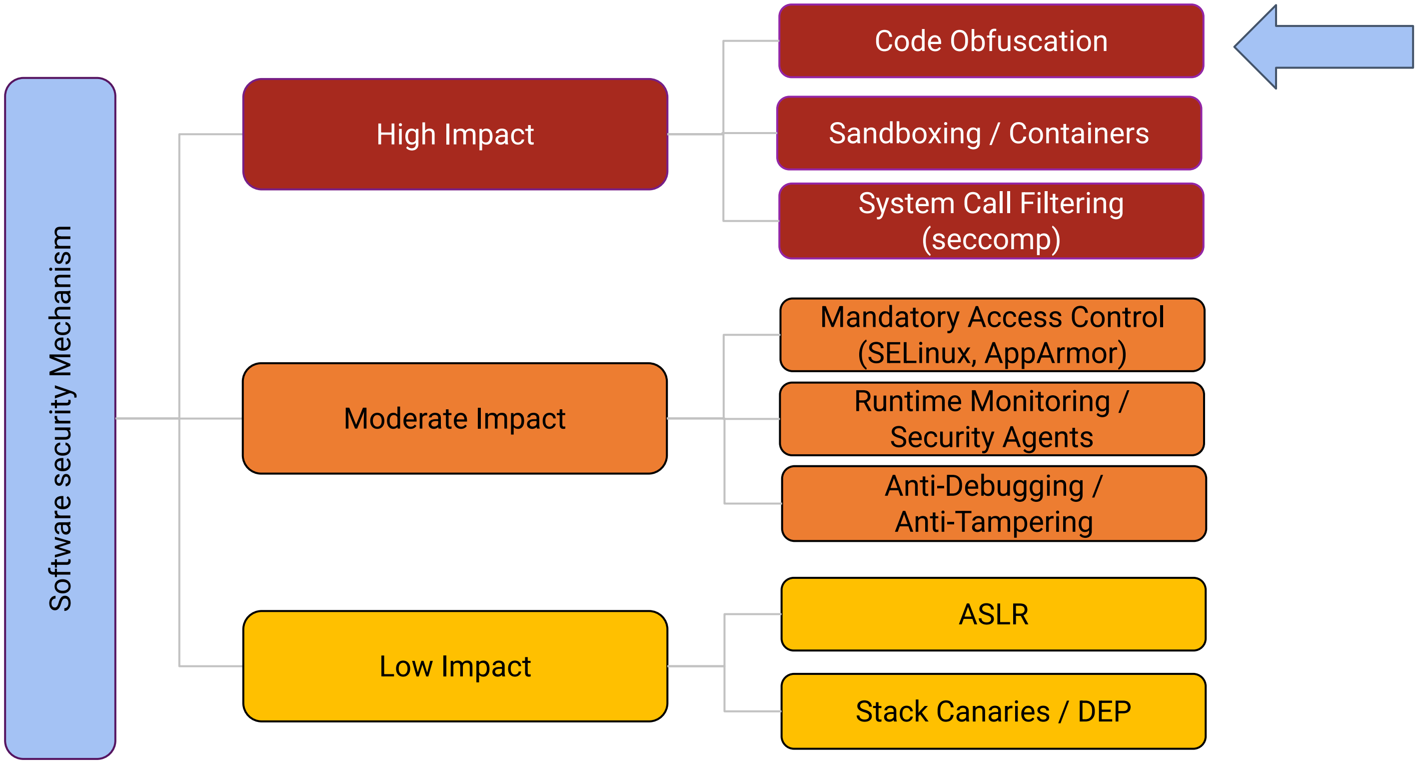
- Ground truth Dataset (7 : critical , 62: noise)
- Last results:
  - TP = 7, FN = 0, FP = 48, TN = 14
  - Precision = Tp/Tp+Fp = 7/(7+48) = 12.7%
  - Recall = Tp/Tp+Fn = 7/(7+0) = 100%
  - F1 = 2(P*R/P+R) = 22.5 %
- updates:
  - TP = 7, FN = 0, FP = 20, TN = 42
  - Precision = 26%
  - Recall = 100%
  - F1 = 41.2 %

# Software Security Mechanisms and Observability

- Security mechanisms can **alter the behavior of software**
- These alterations affect **observable traces**, both at **user level** and **kernel level**
- Downstream applications like **regression detection, anomaly detection, or performance monitoring** rely on traces
- Goal: Understand **how security mechanisms impact trace fidelity and analysis accuracy**

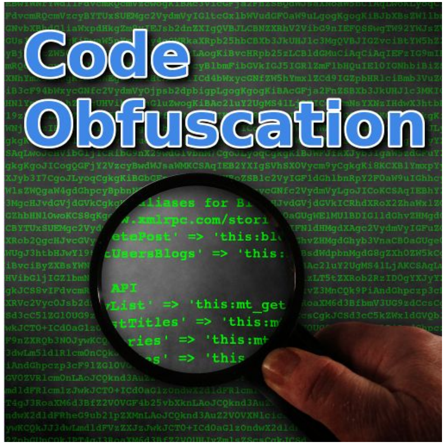


# Software Security Mechanism



# Code Obfuscation

- Protect software by making code
  - **harder to understand**
  - **analyze**
  - **reverse engineer**
  - while preserving functionality.
- Structural Obfuscation
- Control Flow Obfuscation
- Data Obfuscation
- Lexical / Layout Obfuscation
- Name and API Obfuscation

# Evaluation

- Measuring the effects on user and kernel level traces
    - **Defining some measures**
    - **visually (flame graph in trace compass , …)**

- Assessing the effects on a downstreaming task
    - **Regression detection**

# Thank you for your attention!

fateme.faraji-daneshgar@polymtl.ca