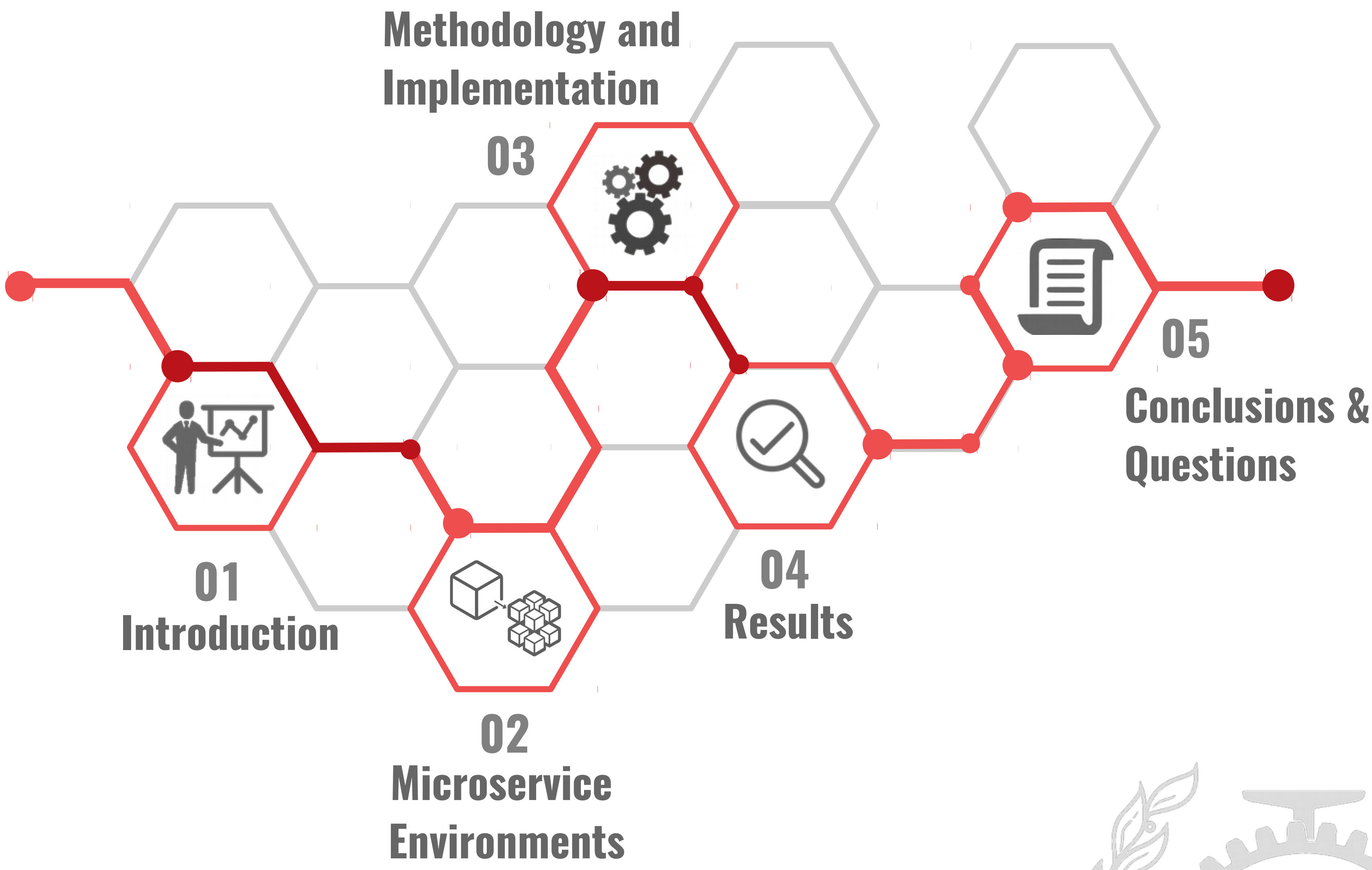
Anomaly detection in microservice systems using tracing data and Machine Learning

Iman Kohyarnejadfard
Prof. Daniel Aloise
Prof. Michel Dagenais
Vahid Azhari

POLYTECHNIQUE MONTREAL

June 2021

Agenda


01
Introduction
02
Microservice Environments
03
Methodology and Implementation
04
Results
05
Conclusions & Questions

Performance Anomaly

Anomalies are patterns in data that do not conform to the expected normal behavior.

Anomalies are the most significant obstacles to the system to perform confidently and predictably.

Sources

application bugs, updates, hardware failure, etc.

performance anomalies are different from high resource consumption.

Workload

the application imposes continuous and more than expected average workload intensity to the system.

Anomaly Detection

Performance anomaly detection refers to the problem of finding exceptional patterns in execution flow that do not conform to the expected normal behavior.

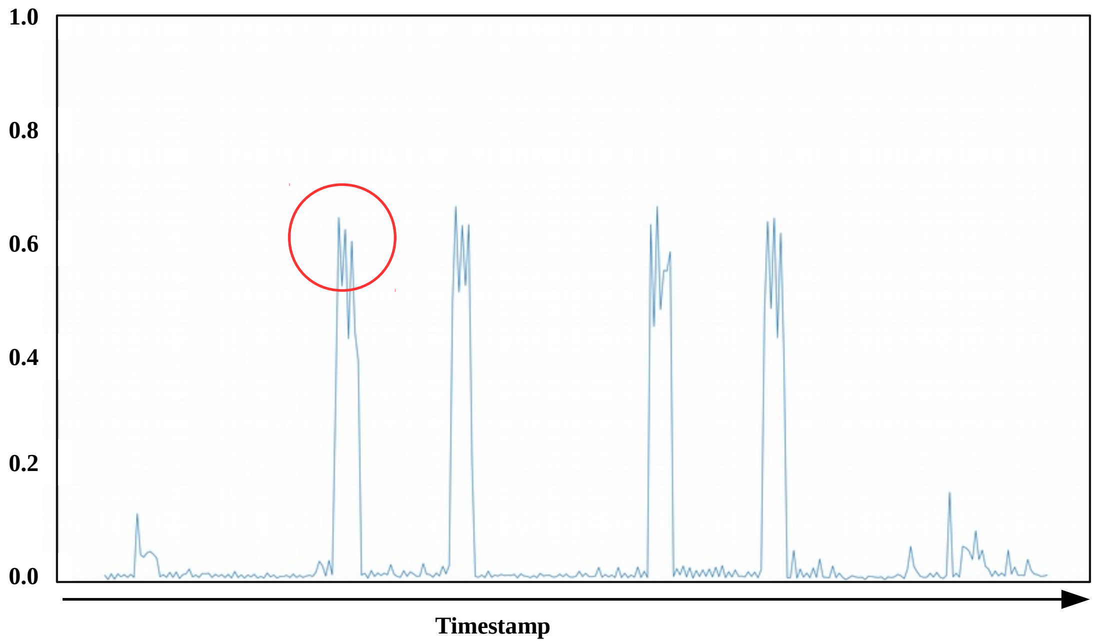
Performance monitoring tools do not provide any details about the applications execution flow.

Anomalies make the execution flow different from the normal situation.

It is an exhausting responsibility for human administrators to manually examine a massive amount of low-level tracing data

Anomaly Detection

This work aims to direct developers to the most relevant problem sites and help them look at just a few small parts instead of the whole trace.



It detects anomalous spans and events during a given trace and speeds up identifying the root cause of the problem through more analysis of the detected anomalous behaviors.

Microservice-based applications vs. traditional applications

01 Microservices are small services that are interconnected with many other microservices to present complex services like web applications; if one service fails, the others will continue to work.

02 Microservices provide greater scalability and make distributing the application over multiple physical or virtual systems possible.

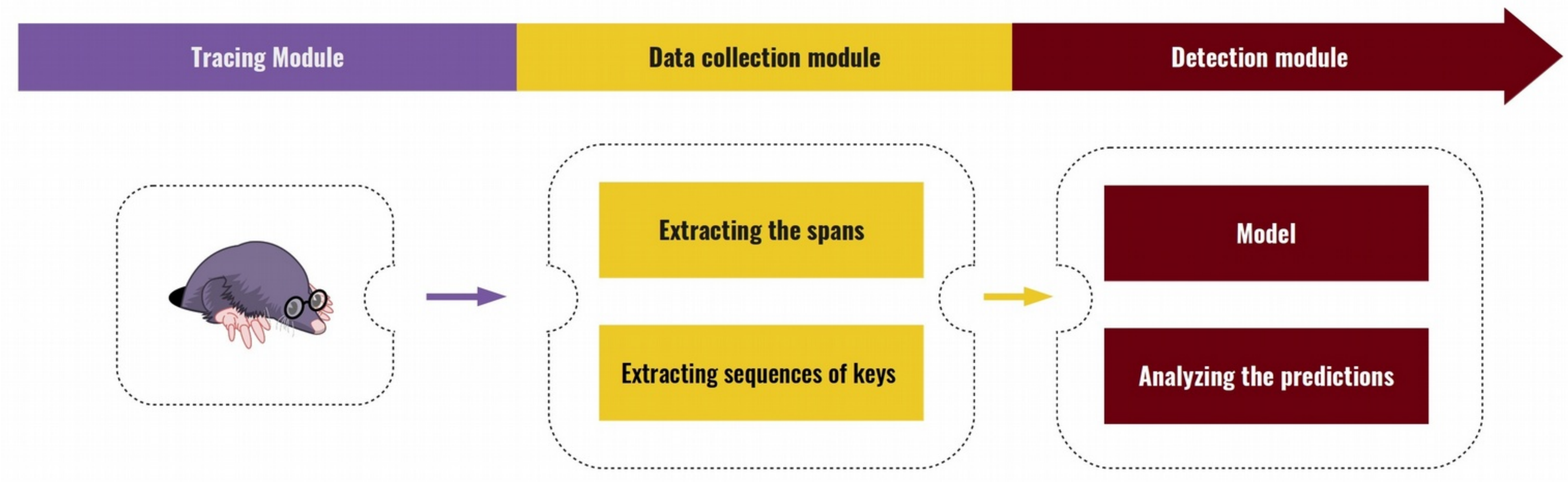
03 Microservices are faster to develop and easier to manage. Moreover, microservice can be written using different technologies.

Despite all these benefits, by increasing the degree of automation and distribution, application performance monitoring becomes more challenging. Microservices may be replaced within seconds, and these changes could also be the cause of anomalies.

Hence, an accurate anomaly detection framework with minimum human intervention is required.

Methodology

- The methodology is based on collecting sequences of events during spans and sending them to the machine learning module.
- The model learns the possible sequence of events and predicts the next event.
- In the detection phase, we use this sequential information to make a prediction and compare the predicted output against the observed value.



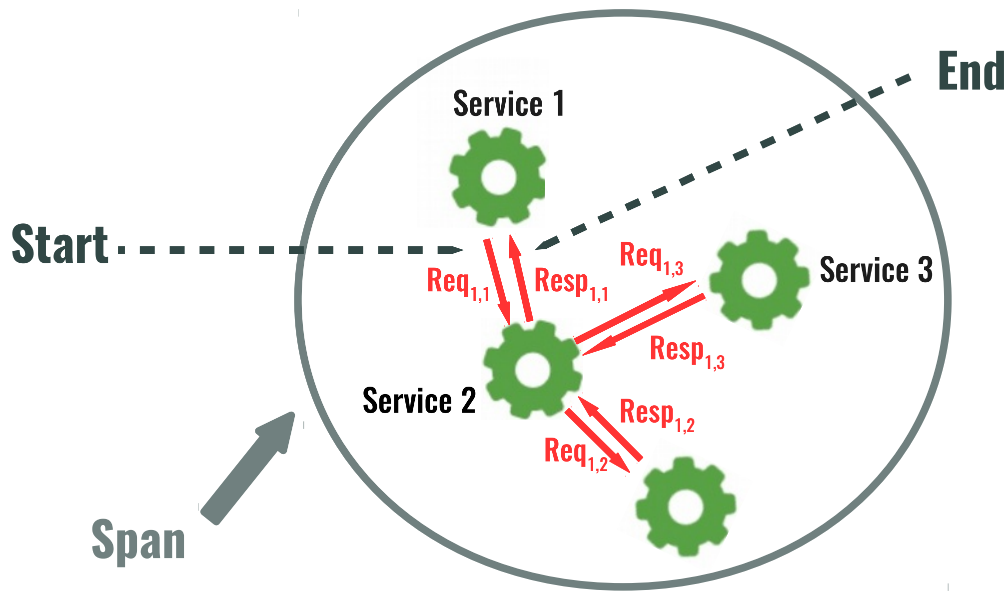
Distributed tracing

A microservice-based application consists of tens, hundreds, or thousands of services running across many hosts, and it is no longer possible to rely on an individual trace.

Distributed tracing provides a view of a request's life as it travels across multiple hosts and services communicating over various protocols.

OpenTracing vs. **LTTng**: Different in the way we collect spans.

The "span" is the primary building block of a distributed trace, representing an individual unit of work done in a distributed system.

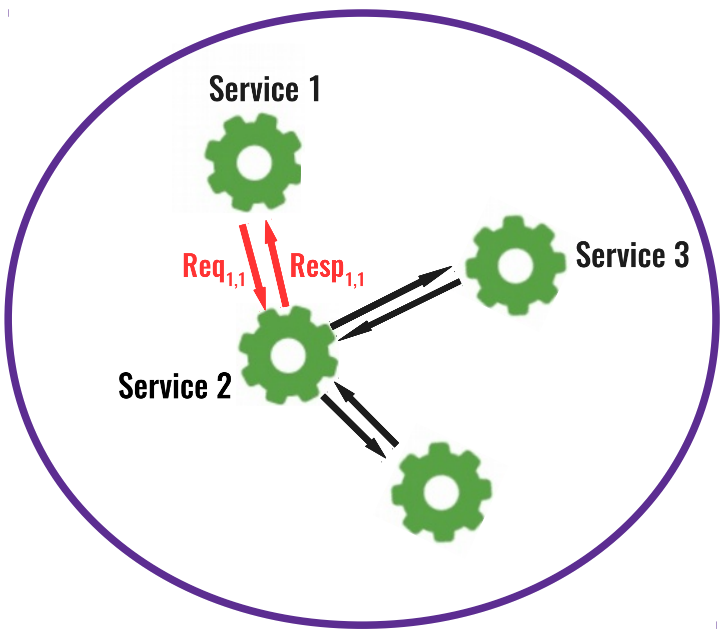


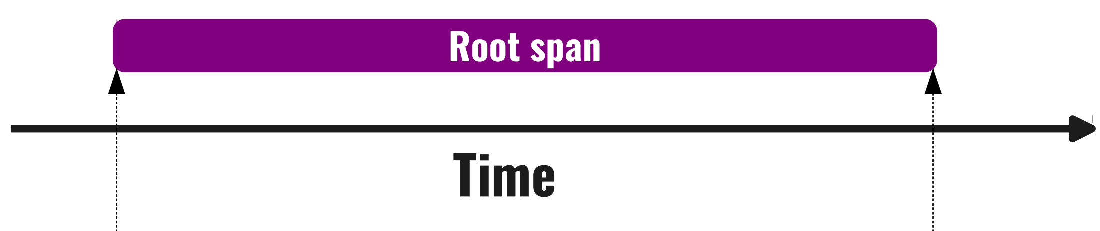
Our Use Case

- We deployed the target microservice environment (developed by Ciena Co.) on a virtualized platform.
- In order to create the training data, 12 traces with the duration of 5 to 10 minutes were obtained from the previous stable releases of the studied software.
- After removing incomplete spans, 61709 spans and 4028 unique keys were extracted.
- The latest release of the software was investigated to evaluate our methodology.
- In test scenarios, significant CPU and disk stress were injected into the nodes.

Extracting spans and subspans

- In the traces we collected from the Ciena simulator, ReqResp events produce spans.
- Each span initiates with a request and ends with a response.
- Requests and responses that happen during a span share a unique tag for example Tag = 00.





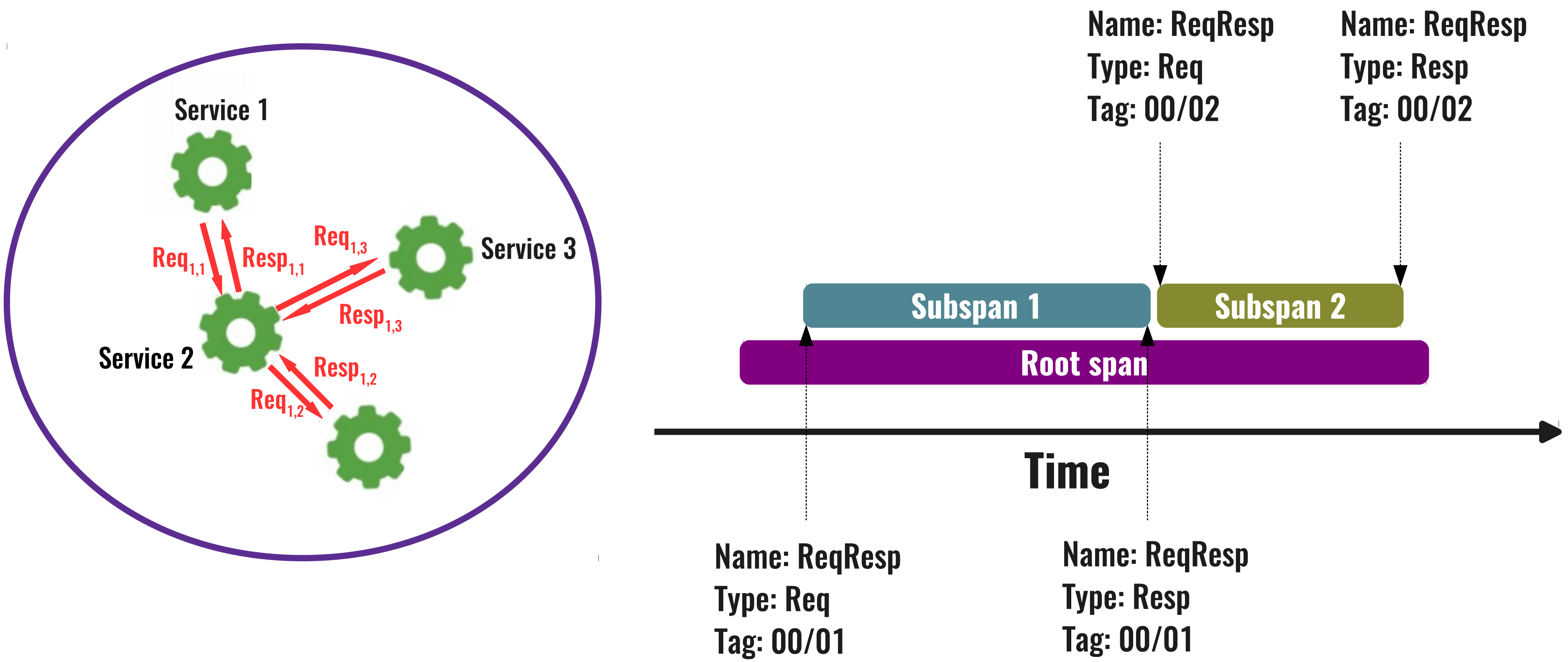
Name: ReqResp
Type: Req
Tag: 00

Name: ReqResp
Type: Resp
Tag: 00

Extracting spans and subspans

Many subspans may be generated during a span's lifetime.

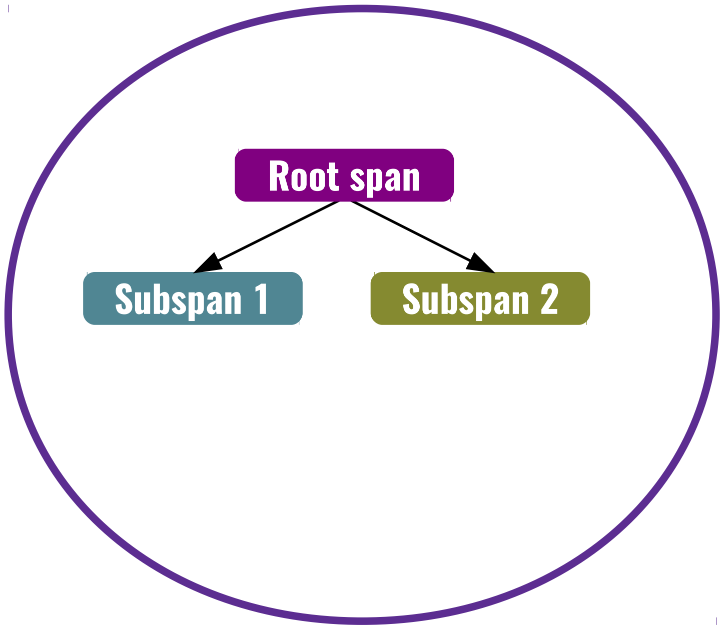
The tag of sub spans parent is embedded in their tag. For example, “00/01” indicates the span “00” is the parent of sub span “01”.

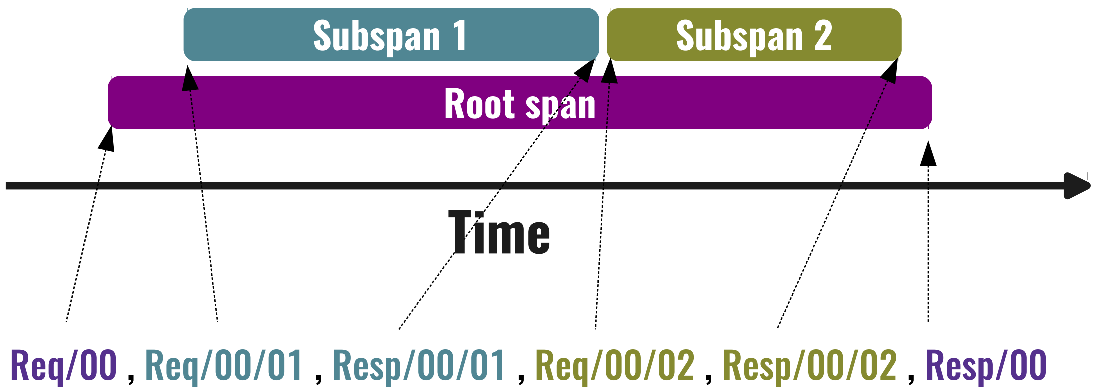


Extracting spans and subspans

Many subspans may be generated during a span's lifetime.

The tag of sub spans parent is embedded in their tag. For example, “00/01” indicates the span “00” is the parent of sub span “01”.





Using arguments and generating keys

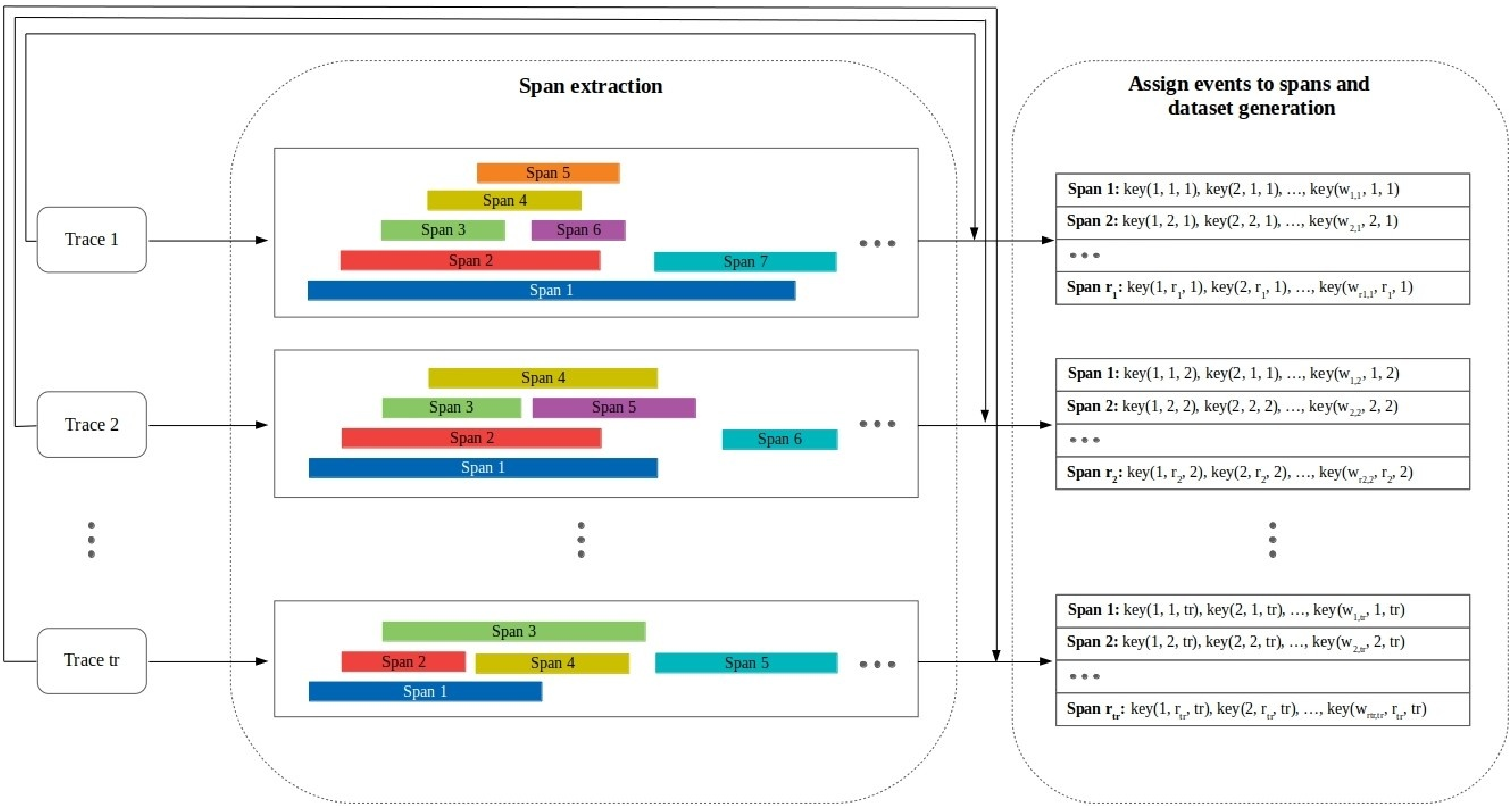
Tens of userspace and kernel events happen during spans. Therefore, we put these events in the right place in the sequence.

The scope of this work is limited to the arguments that are common to all events.

We concatenate the arguments of an event and provide a single event representation (key).

Event category	Arguments	Type
Request/Response	Name	string
	Type	string
	Tag	string
	Procname	string
Other	Name	string
	Procname	string
	Message	string

Dataset

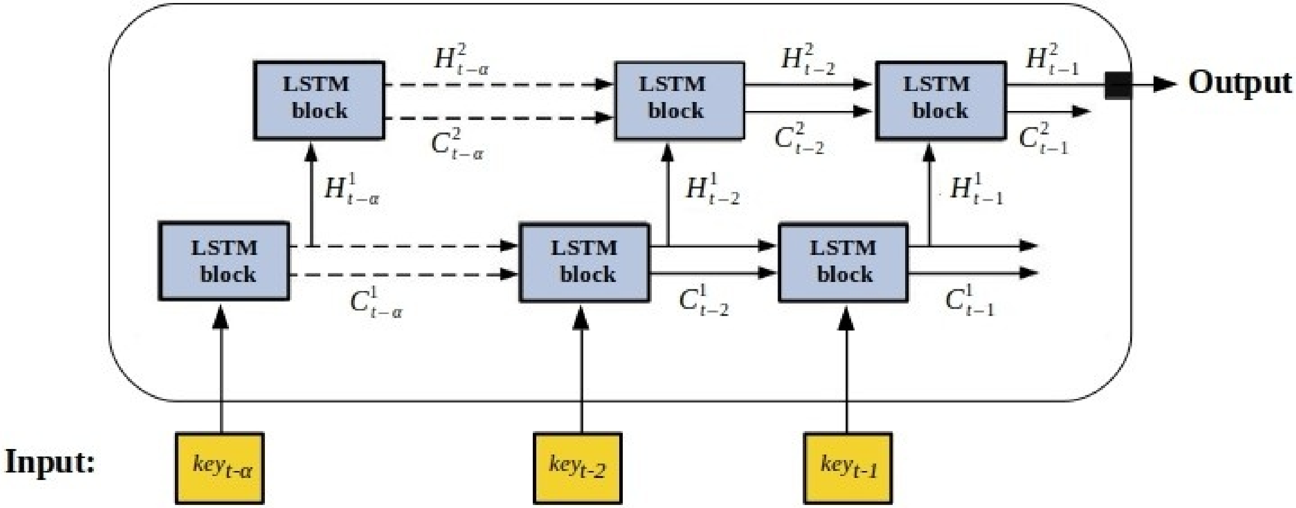

Span extraction
Assign events to spans and dataset generation
Trace 1
Trace 2
Trace tr
Span 5
Span 4
Span 3
Span 6
Span 2
Span 7
Span 1
Span 4
Span 3
Span 5
Span 2
Span 6
Span 1
Span 3
Span 2
Span 4
Span 5
Span 1
Span 1: key(1, 1, 1), key(2, 1, 1), …, key($w_{1,1}$, 1, 1)
Span 2: key(1, 2, 1), key(2, 2, 1), …, key($w_{2,1}$, 2, 1)
Span r_1: key(1, r_1, 1), key(2, r_1, 1), …, key($w_{r1,1}$, r_1, 1)
Span 1: key(1, 1, 2), key(2, 1, 1), …, key($w_{1,2}$, 1, 2)
Span 2: key(1, 2, 2), key(2, 2, 2), …, key($w_{2,2}$, 2, 2)
Span r_2: key(1, r_2, 2), key(2, r_2, 2), …, key($w_{r2,2}$, r_2, 2)
Span 1: key(1, 1, tr), key(2, 1, tr), …, key($w_{1,tr}$, 1, tr)
Span 2: key(1, 2, tr), key(2, 2, tr), …, key($w_{2,tr}$, 2, tr)
Span r_{tr}: key(1, r_{tr}, tr), key(2, r_{tr}, tr), …, key($w_{rtr,tr}$, r_{tr}, tr)

Detection Module

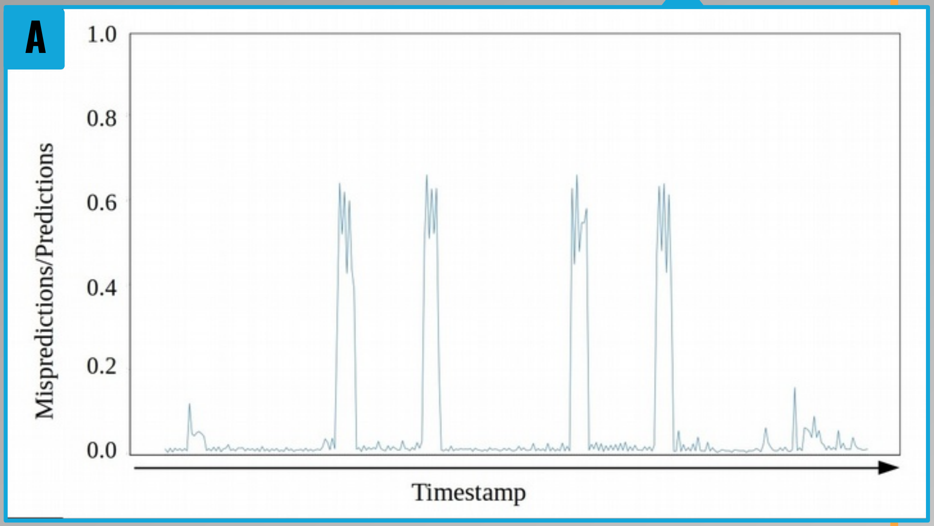
- A limited number of events can be the result of an action. Therefore few of the possible events can appear as the next event in the sequence.
- The fundamental intuition behind this work is natural language processing.
- As elements of a sequence, events follow specific patterns and grammar rules.
- We used an LSTM neural network to learn a model of event patterns from normal execution.

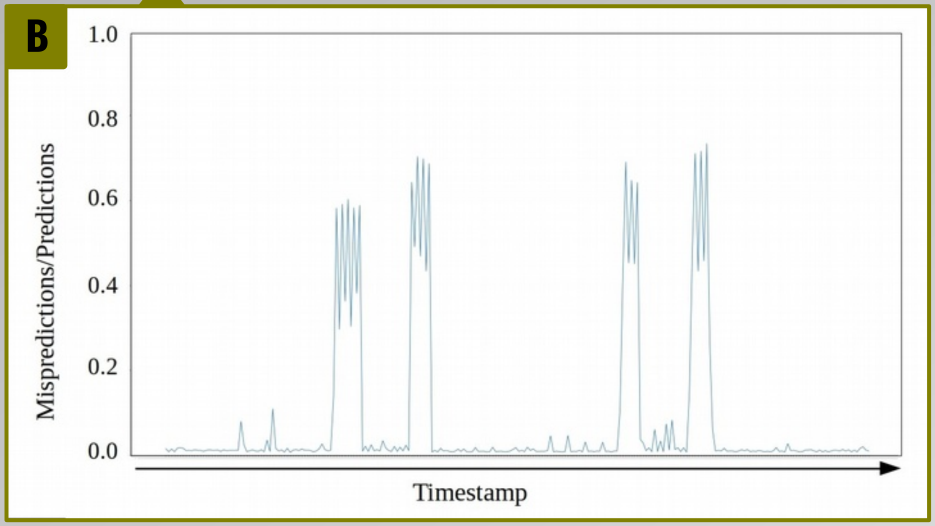
Detection Module

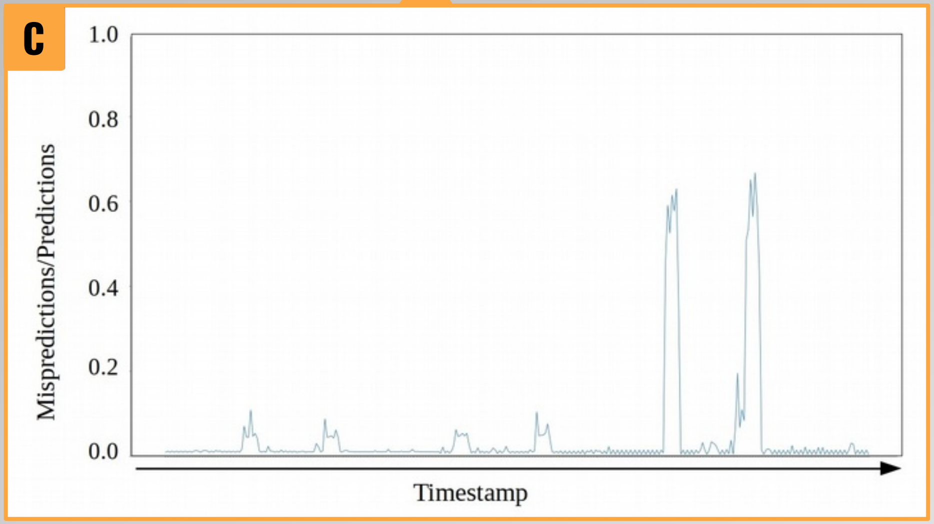
- The model learns a probability distribution $Prob(key_t=v_i|\{key_{t-\alpha}, key_{t-(\alpha-1)}, ..., key_{t-1}\})$ that maximizes the probability of the training sequence.
- In prediction phase the model marks key_t as a correct predicted event if the probability of the observed key_t is bigger than 0.5.
- Otherwise, that event is flagged as misprediction.
- Spans in which mispredictions occur frequently are classified as anomalous spans.



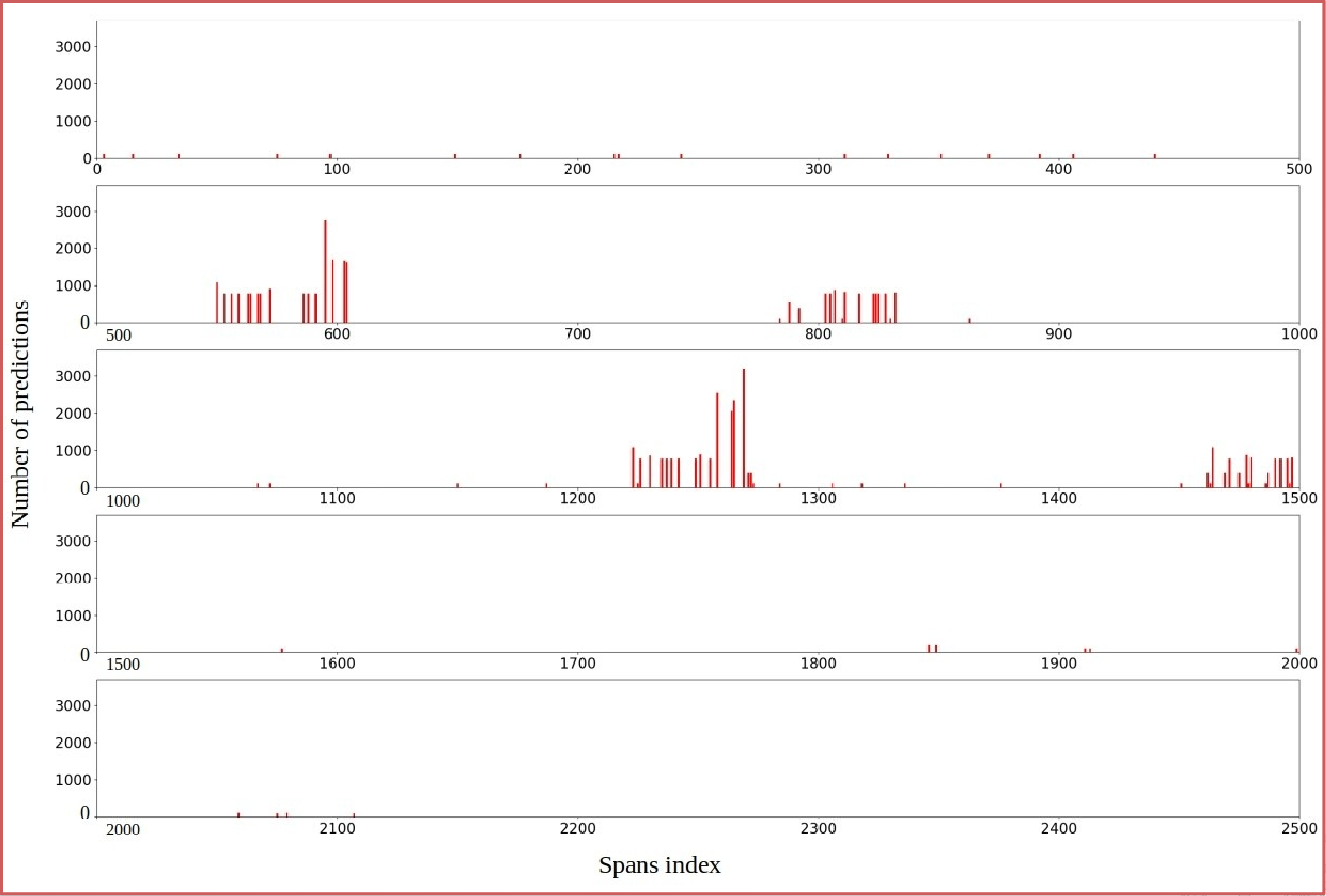
RESULTS


A
1.0
0.8
0.6
0.4
0.2
0.0
Mispredictions/Predictions
Timestamp

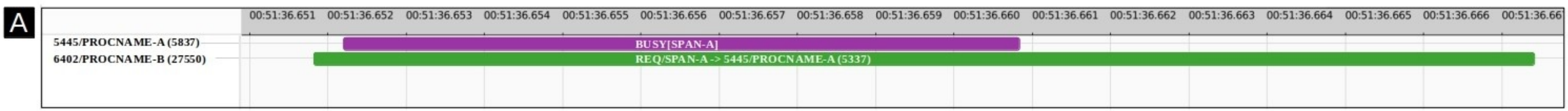

B
1.0
0.8
0.6
0.4
0.2
0.0
Mispredictions/Predictions
Timestamp

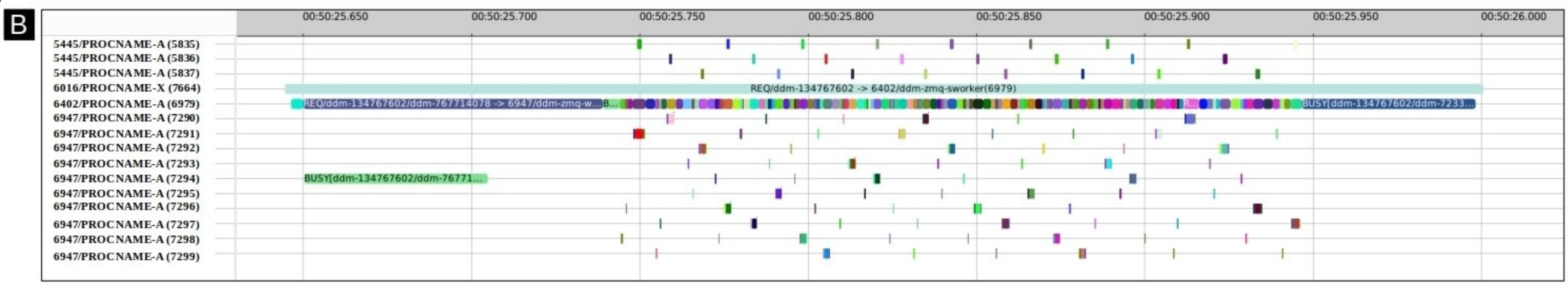

C
1.0
0.8
0.6
0.4
0.2
0.0
Mispredictions/Predictions
Timestamp

R E S U L T S


Number of predictions
Spans index
3000
2000
1000
0
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
2400
2500

Results


A
00:51:36.651
00:51:36.652
00:51:36.653
00:51:36.654
00:51:36.655
00:51:36.656
00:51:36.657
00:51:36.658
00:51:36.659
00:51:36.660
00:51:36.661
00:51:36.662
00:51:36.663
00:51:36.664
00:51:36.665
00:51:36.666
5445/PROCNAME-A (5837)
6402/PROCNAME-B (27550)
BUSY[SPAN-A]
REQ/SPAN-A -> 5445/PROCNAME-A (5337)


B
00:50:25.650
00:50:25.700
00:50:25.750
00:50:25.800
00:50:25.850
00:50:25.900
00:50:25.950
00:50:26.000
5445/PROCNAME-A (5835)
5445/PROCNAME-A (5836)
5445/PROCNAME-A (5837)
6016/PROCNAME-X (7664)
6402/PROCNAME-A (6979)
6947/PROCNAME-A (7290)
6947/PROCNAME-A (7291)
6947/PROCNAME-A (7292)
6947/PROCNAME-A (7293)
6947/PROCNAME-A (7294)
6947/PROCNAME-A (7295)
6947/PROCNAME-A (7296)
6947/PROCNAME-A (7297)
6947/PROCNAME-A (7298)
6947/PROCNAME-A (7299)
REQ/ddm-134767602 -> 6402/ddm-zmq-sworker(6979)
REQ/ddm-134767602/ddm-767714078 -> 6947/ddm-zmq-w...
BUSY[ddm-134767602/ddm-7233...
BUSY[ddm-134767602/ddm-76771...

Thank you for your attention!

Questions?

Iman.kohyarnejadfard@polymtl.ca
https://github.com/kohyar

References

[1] Z. Xu, X. Yu, Y. Feng, J. Hu, Z. Tari, and F. Han. A multi-module anomaly detection scheme based on system call prediction. In 2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA), pages 1376–1381, June 2013.

[2] A. Liu, C. Martin, T. Hetherington, and S. Matzner. A comparison of system call feature representations for insider threat detection. In Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop, pages 340–347, June 2005.

[3] Michael Dymshits, Ben Myara, and David Tolpin. Process monitoring on sequences of system call count vectors. 2017 International Carnahan Conference on Security Technology (ICCST), pages 1–5, 2017.

[4] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In Byeong Ho Kang and Quan Bai, editors, AI 2016: Advances in Artificial Intelligence, pages 137–149, Cham, 2016. Springer Iternational Publishing.

[5] Mathieu Desnoyers and Michel Dagenais. The lttng tracer : A low impact performance and behavior monitor for gnu / linux. In OLS Ottawa Linux Symposium, 2006.

[6] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg, 2006.

[7] Ulrich H.-G. Kre. Advances in kernel methods. chapter Pairwise Classification and Support Vector Machines, pages 255–268. MIT Press, Cambridge, MA, USA, 1999.

[8] Du, Qingfeng, Tiandi Xie, and Yu He. "Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring." International Conference on Algorithms and Architectures for Parallel Processing. Springer, Cham, 2018.

[9] Cao, Wei, Zhiying Cao, and Xiuguo Zhang. "Research on Microservice Anomaly Detection Technology Based on Conditional Random Field." Journal of Physics: Conference Series. Vol. 1213. No. 4. IOP Publishing, 2019.

[10] Nikiforov, Roman. "Clustering-based Anomaly Detection for microservices." arXiv preprint arXiv:1810.02762 (2018).

[11] Pahl, Marc-Oliver, and François-Xavier Aubet. "All eyes on you: Distributed Multi-Dimensional IoT microservice anomaly detection." 2018 14th International Conference on Network and Service Management (CNSM). IEEE, 2018.

[12] Nandi, Animesh, et al. "Anomaly detection using program control flow graph mining from execution logs." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016.