25+ years of Software Performance: from integrated system modelling to AI-based runtime analysis… any relation to sustainability?

VITTORIO CORTELLESSA

Information Engineering, Computer Science, and Mathematics Dept.

Università dell’Aquila

vittorio.cortellessa@univaq.it

1st International Workshop on Systems and Methods for Sustainable Large-Scale AI (GreenSys) @ EuroSys 2025, Rotterdam (NL)

Credits for this presentation

Daniele Di Pompeo

Michele Tucci

Luca Traini

Federico Di Menna

Roadmap

- Introduction

- Performance modeling
- Results interpretation

- Performance at execution time (using AI)

… any relation to sustainability?!

- Conclusions

Introduction

At the roots of Software Performance

Since early 70s…

System Performance

End of 90s…

… system splitting

in performance “contributors”…

user

software

hardware

At the roots of Software Performance

software

user

User invokes software services

Software services use hardware resources

hardware

Software as a first-class artifacts in performance assessment of computer systems

At the roots of Software Performance

Performance parameters

user

Workload

Number of requests per unit of time

Operational profile

Probability of execution of "some part" of the software

software

Service demand

Amount of resources needed to software execution

hardware

Service rate

Number of operations completed by each resource per unit of time

At the roots of Software Performance

user

software

Easier experimentation of different SW solutions on the same HW platform (and viceversa)

hardware

Not only hardware bottlenecks to remove, but also software refactoring solutions

More knobs in the hands of performance analyzers!

Software performance : looking at last 25+ years

- Performance model generation:
  languages and transformations
- Analysis results interpretation:
  performance antipatterns
- Performance analysis at system execution time

Model

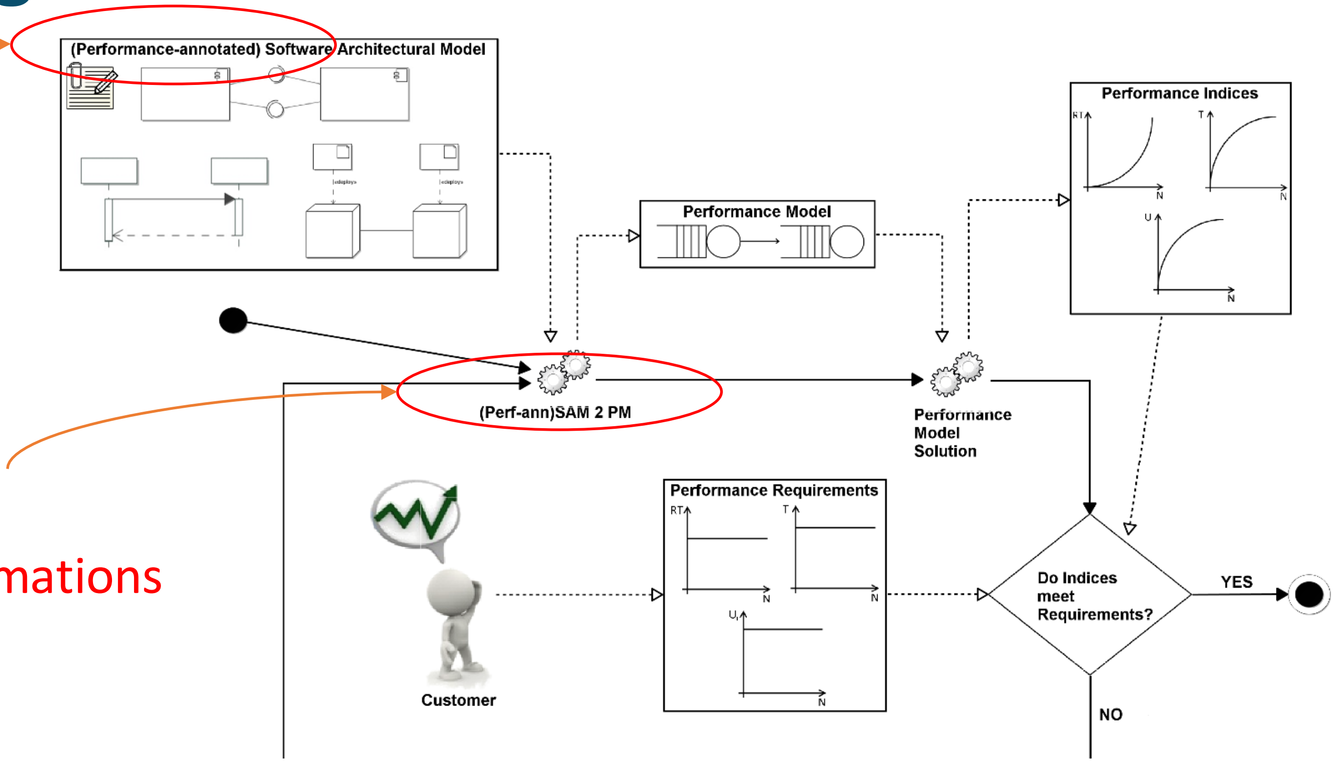
Code

Performance model generation: languages and transformations

Software Performance Engineering : model generation

Augmented modeling languages

Model transformations



*Miguel de Miguel, Thomas Lambolais, Mehdi Hannouz, Stéphane Betgé-Brezetz, Sophie Piekarec: **UML extensions for the specification and evaluation of latency constraints in architectural models.*** WOSP 2000

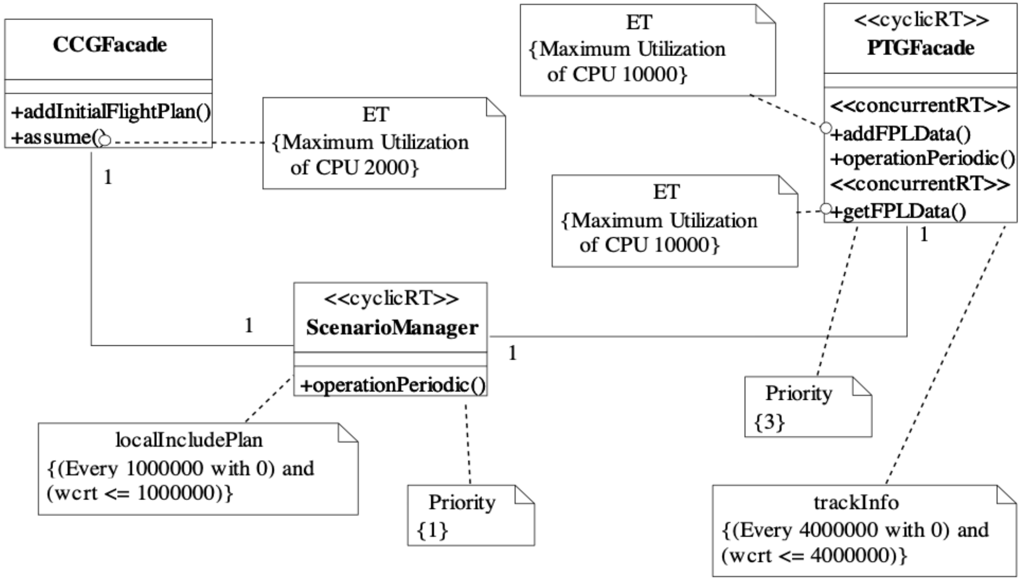


Figure 1. Application of UML extensions in a static diagram.

UML extensions: multiview aspects

C. Murray Woodside, Dorina C. Petriu, Dorin Bogdan Petriu, Hui Shen, Toqeer Israr, José Merseguer: ***Performance by unified model analysis (PUMA).***

WOSP 2005

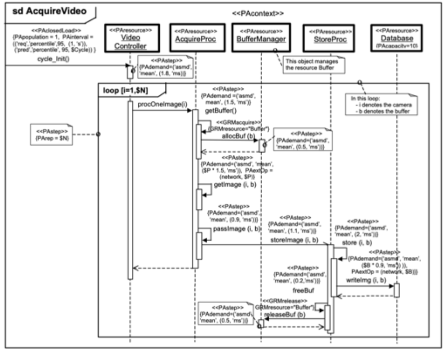


Figure 4. A UML2 Interaction Diagram for the Acquire/Store Video scenario for the Building Security System from [15]

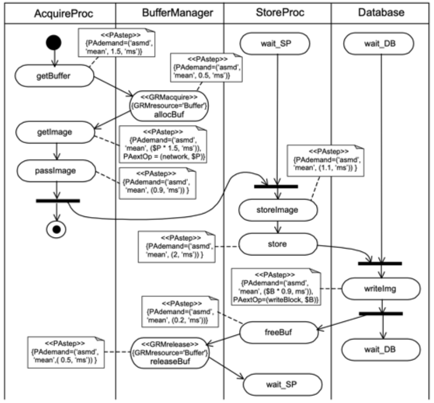


Figure 3 UML1.4 Activity Diagram for the Acquire/Store Video Scenario for the building security system

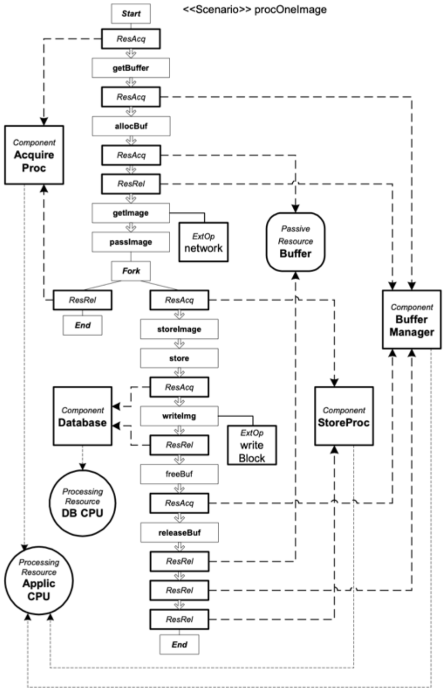
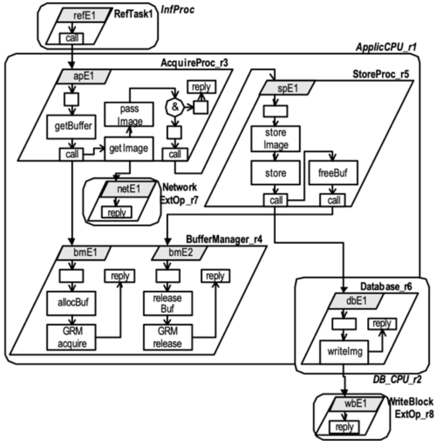


Figure 5 Core Scenario Model for the Video Scenario.



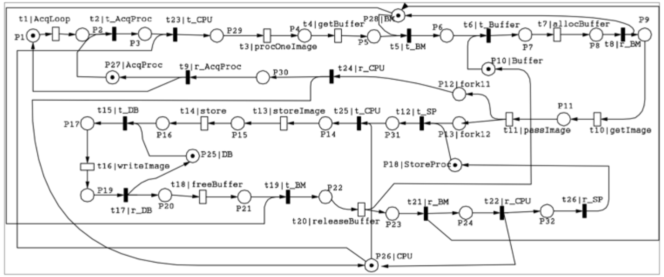


Figure 12. Petri Net produced automatically for the Video Acquisition scenario

Pivot language: multiple sources to multiple targets

… any relationship to sustainability?

- Separation of **software** and **hardware** for sustainability **modeling**
- Modeling languages for **representing sustainability metrics**

Our contribution to model-based performance/sustainability joint analysis

2024 IEEE 21st International Conference on Software Architecture (ICSA)

Exploring sustainable alternatives for the deployment of microservices architectures in the cloud

Vittorio Cortellessa
SPENCER Lab
University of L'Aquila, Italy
vittorio.cortellessa@univaq.it

Daniele Di Pompeo
SPENCER Lab
University of L'Aquila, Italy
daniele.dipompeo@univaq.it

Michele Tucci
SPENCER Lab
University of L'Aquila, Italy
michele.tucci@univaq.it

***Abstract*—As organizations increasingly migrate their applications to the cloud, the optimization of microservices architectures becomes imperative for achieving sustainability goals. Nonetheless, sustainable deployments may increase costs and deteriorate performance, thus the identification of optimal trade-offs among these conflicting requirements is a key objective not easy to achieve. This paper introduces a novel approach to support cloud deployment of microservices architectures by targeting optimal combinations of application performance, deployment costs, and power consumption. By leveraging genetic algorithms, specifically NSGA-II, we automate the generation of alternative architectural deployments. The results demonstrate the potential of our approach through a comprehensive assessment of the Train Ticket case study.**

***Index Terms*—sustainability, refactoring, performance, search-based software engineering, model-driven engineering**

I. Introduction

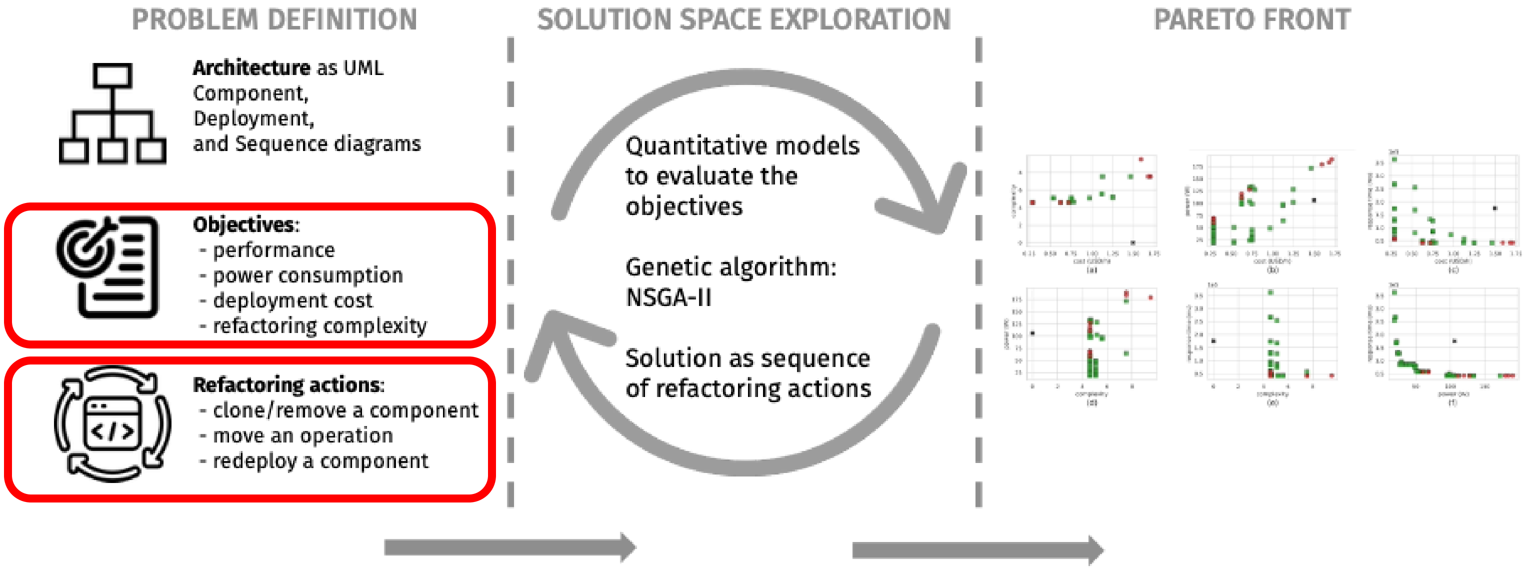
ensure the long-term viability of cloud-based microservices architectures.

A number of approaches [9]–[11] emerged in recent years to optimize energy consumption and cost when deploying to the cloud. Nevertheless, these approaches seldom address this problem at the architectural level and, as a consequence, often they lack the capability to empower designers with a comprehensive understanding of the intricate trade-offs emerging from this task. This lack of understanding is further exacerbated by the fact that the energy consumption of a microservices architecture is not only a function of the deployment configuration but also of the user behavior [12].

In this paper, we aim to address this lack by presenting a novel approach to explore sustainable solutions when deploying microservices architectures in the cloud. Specifically, we exploit NSGA-II [13] to generate diverse deployment

V. Cortellessa, D. Di Pompeo, M. Tucci

Exploring Sustainable Alternatives for the Deployment of Microservices Architectures in the Cloud *(ICSA 2024)*



Exploring Sustainable Alternatives for the Deployment of Microservices Architectures in the Cloud *(ICSA 2024)*

Objectives

Deployment cost

Sum of the hourly cost of all the used nodes.

Performance

Sum of the average response times of all the types of requests.

Refactoring complexity

Estimation of the effort required to refactor the architecture.

$$complexity = \sum_{a \in S} C_{base}(a) \cdot C_{arch}(a, e)$$

Power consumption

Total power of a cloud instance by combining active and scaled-down idle CPU power.

V. Cortellessa, D. Di Pompeo, M. Tucci

Exploring Sustainable Alternatives for the Deployment of Microservices Architectures in the Cloud *(ICSA 2024)*

Refactoring actions

Redeploy Existing Component

Move a component to a new node while maintaining its connections to all originally linked nodes.

Clone Node

Create a replica of the node with its components and connections.

Remove Node

Remove a node and relocate its components to other nodes.

Relocate Operation to Existing Component

Move an operation to a different (existing) component.

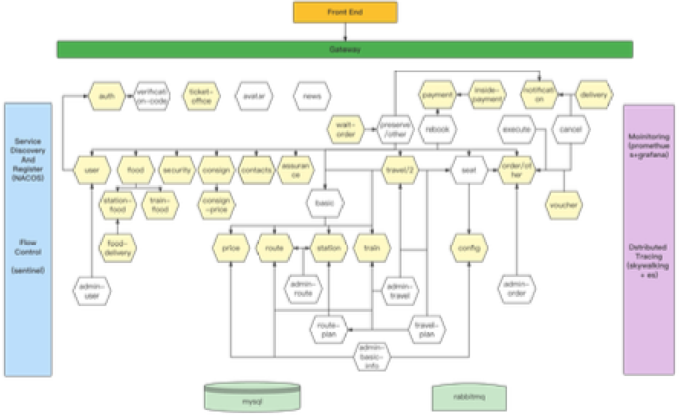
Move Operation to New Component on New Node

Complex action that creates a new node and component to host an operation.

V. Cortellessa, D. Di Pompeo, M. Tucci

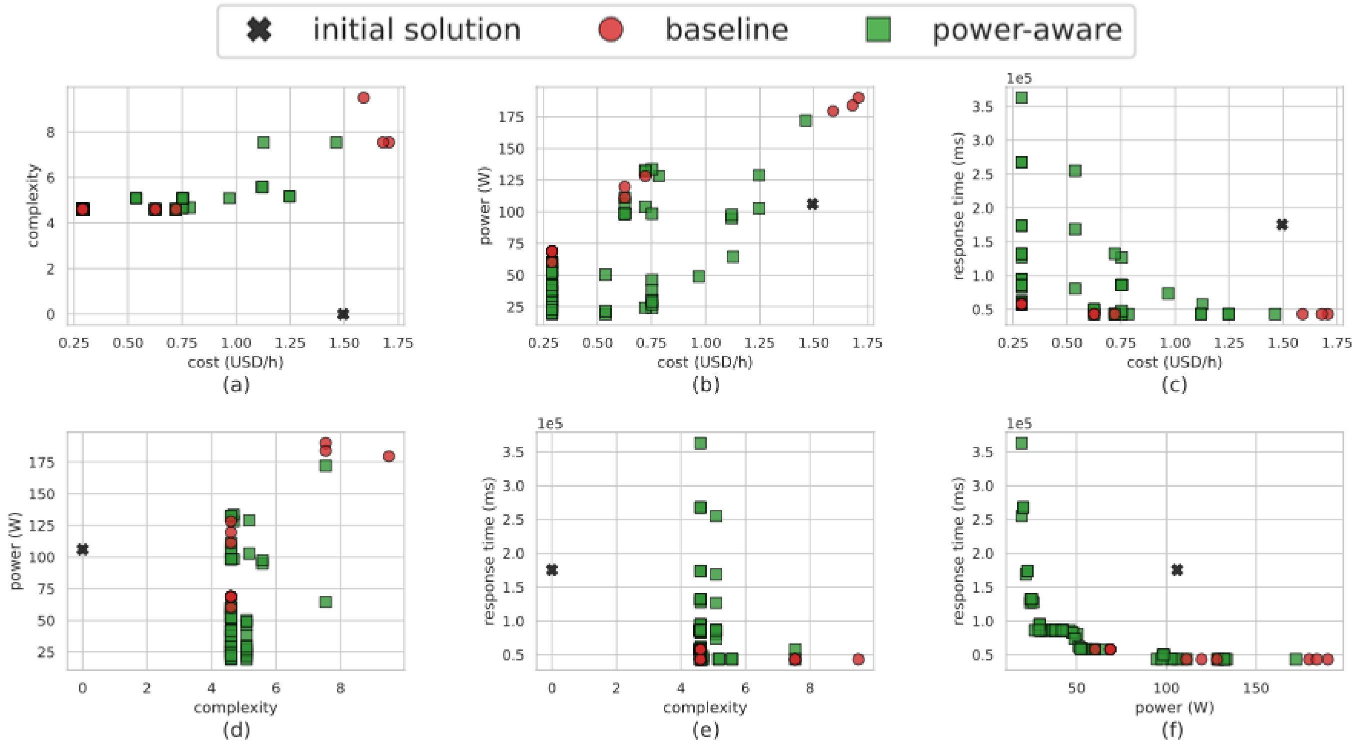
Exploring Sustainable Alternatives for the Deployment of Microservices Architectures in the Cloud *(ICSA 2024)*

- Approach tested on **Train Ticket Booking Service (TTBS)**, a web-based application with **40 microservices**.
- On **three scenarios**: Login, Update user details, and Rebook a ticket.
- We **reverse engineered** the architectural specification, including component, deployment, and sequence diagrams (in UML format).



V. Cortellessa, D. Di Pompeo, M. Tucci

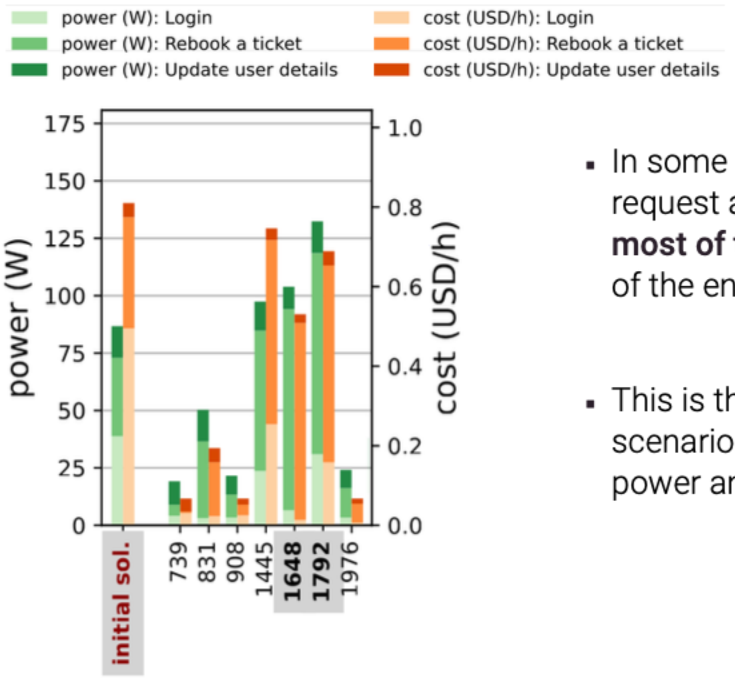
Exploring Sustainable Alternatives for the Deployment of Microservices Architectures in the Cloud *(ICSA 2024)*



V. Cortellessa, D. Di Pompeo, M. Tucci

Exploring Sustainable Alternatives for the Deployment of Microservices Architectures in the Cloud *(ICSA 2024)*

User profiles



- In some configurations, a single type of request appears to be responsible for **most of the power consumption and cost** of the entire system.
- This is the case for the **Rebook a ticket** scenario in solutions with higher values of power and cost.

V. Cortellessa, D. Di Pompeo, M. Tucci

Exploring Sustainable Alternatives for the Deployment of Microservices Architectures in the Cloud *(ICSA 2024)*

Frequencies of types of refactoring actions

Type	Target (N,C,O)	To (N,C)	Frequency baseline	Frequency power-aware
DROP	(N) verification	–	25.00% (13)	24.26% (66)
DROP	(N) login	–	21.15% (11)	21.69% (59)
DROP	(N) order-other	–	19.23% (10)	24.63% (67)
DROP	(N) route-plan	–	13.46% (7)	15.81% (43)
REDO	(C) order-other	(N) new-node	7.69% (4)	0.74% (2)
DROP	(N) travel-plan	–	3.85% (2)	2.57% (7)
MOVE	(O) login	(C) ticket-info	1.92% (1)	0.37% (1)
MOVE	(O) updateuser	(C) travel-plan	1.92% (1)	–
DROP	(N) rebook	–	1.92% (1)	–
DROP	(N) sso	–	1.92% (1)	–
DROP	(N) ticket-info	–	1.92% (1)	0.74% (2)
MOVE	(O) login	(C) verification	–	1.47% (4)
CLON	(N) login	–	–	1.47% (4)
MOVE	(O) getbyid	(C) rebook	–	1.47% (4)
MOVE	(O) login	(C) rebook	–	1.10% (3)
DROP	(N) seat	–	–	1.10% (3)
MOVE	(O) login	(C) travel-plan	–	0.74% (2)
MOVE	(O) rebook	(C) order-other	–	0.74% (2)
CLON	(N) ticket-info	–	–	0.37% (1)
MOVE	(O) login	(C) sso	–	0.37% (1)
MOVE	(O) modify	(C) station	–	0.37% (1)

V. Cortellessa, D. Di Pompeo, M. Tucci

Exploring Sustainable Alternatives for the Deployment of Microservices Architectures in the Cloud *(ICSA 2024)*

What changes when we consider the power consumption?

Type	Target (N,C,O)	To (N,C)	Frequency	
			baseline	power-aware
REDO	(C) order-other	(N) new-node	7.69% (4)	0.74% (2)

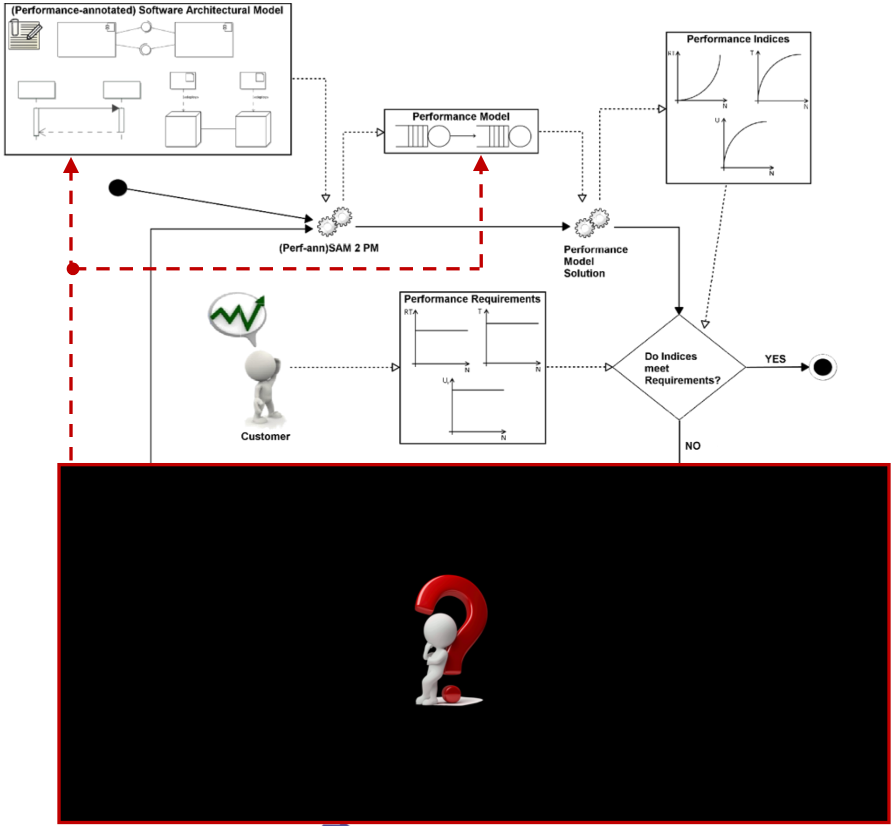
- The **component redeployment action** (REDO) is more frequent in the *baseline* experiment, adding new nodes. In contrast, focusing on *power consumption* leads to node removal, **even at the cost of performance**.
- The MOTN action, **moving an operation to a new component on a new node**, is **absent** in the super Pareto front of both experiments, likely due to the **complexity of creating new nodes**.

… any relationship to sustainability?

- Separation of **software** and **hardware** for sustainability **modeling**
- Modeling languages for **representing sustainability metrics**
- **Refactoring impact** on sustainability

Analysis results interpretation: Performance Antipatterns

Software Performance Engineering : results interpretation



How to (automatically) interpret negative analysis results?

How to generate corrective actions on the model???

Antipatterns : Negative features of a (software) system

- Conceptually **similar to design patterns**: recurring solutions to common design problems
- The definition includes common **mistakes** (i.e. bad practices) in software development as well as their **solutions**

» What to avoid and how to solve (performance) problems!

W.J.Brown, R.C. Malveau, H.W. Mc Cornich III, and T.J. Mowbray.
"Antipatterns: Refactoring Software, Architectures, and Project in Crisis", 1998.

Performance Antipatterns: an example

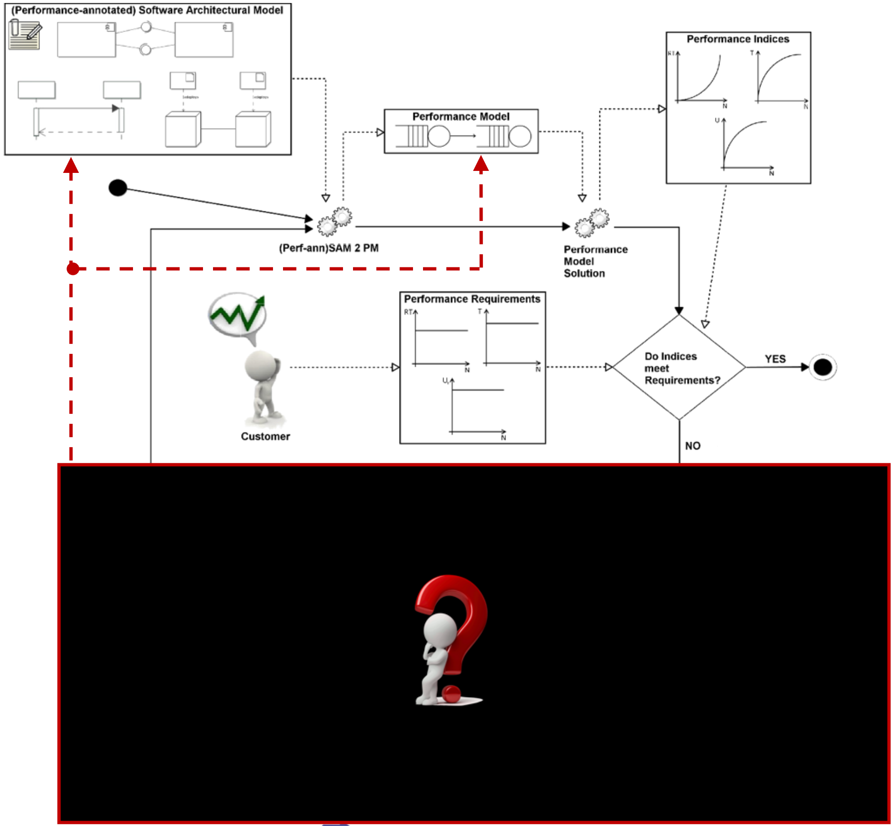
Antipattern	Problem	Solution
Blob	Occurs when a single class or component either 1) performs all of the work of an application or 2) holds all of the applications data. Either manifestation results in excessive message traffic that can degrade performance.	Refactor the design to distibute intelligence uniformly over the applications top-level classes, and to keep related data and behavior together.
…	…	…

They are very complex mostly due to the **different "nature" of their basic elements**

However, they have represented a rich, solid knowledge repository for interpreting analysis results

C. U. Smith and L. G.Williams. "More new software performance antipatterns: Even more ways to shoot yourself in the foot", 2003.

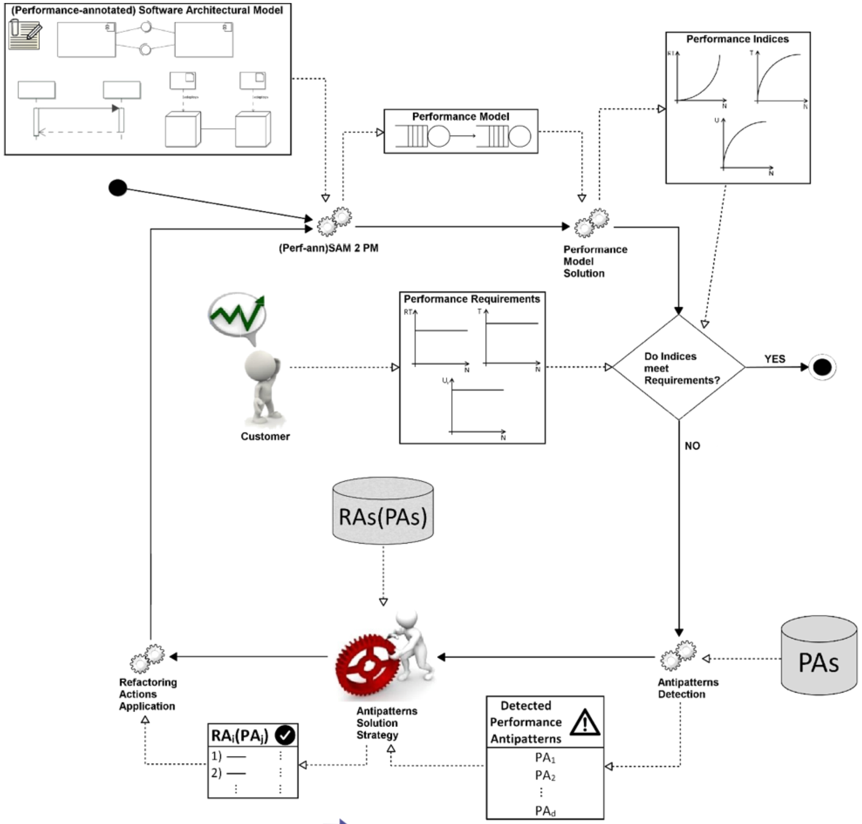
Software Performance Engineering : results interpretation



How to (automatically) interpret negative analysis results?

How to generate corrective actions on the model???

Software Performance Engineering : results interpretation

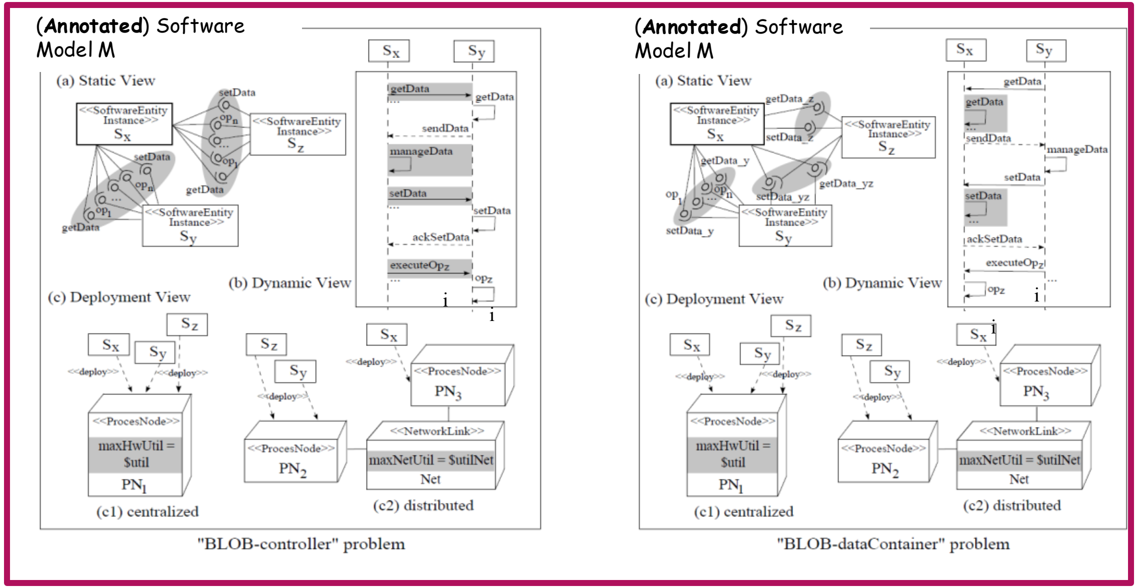


The concept of
Performance Antipattern

Performance Antipattern Representation

PROBLEM: *"It occurs when a single class or component either 1) performs all of the work of an application or 2) holds all of the applications data. Either manifestation results in excessive message traffic that can degrade performance"*

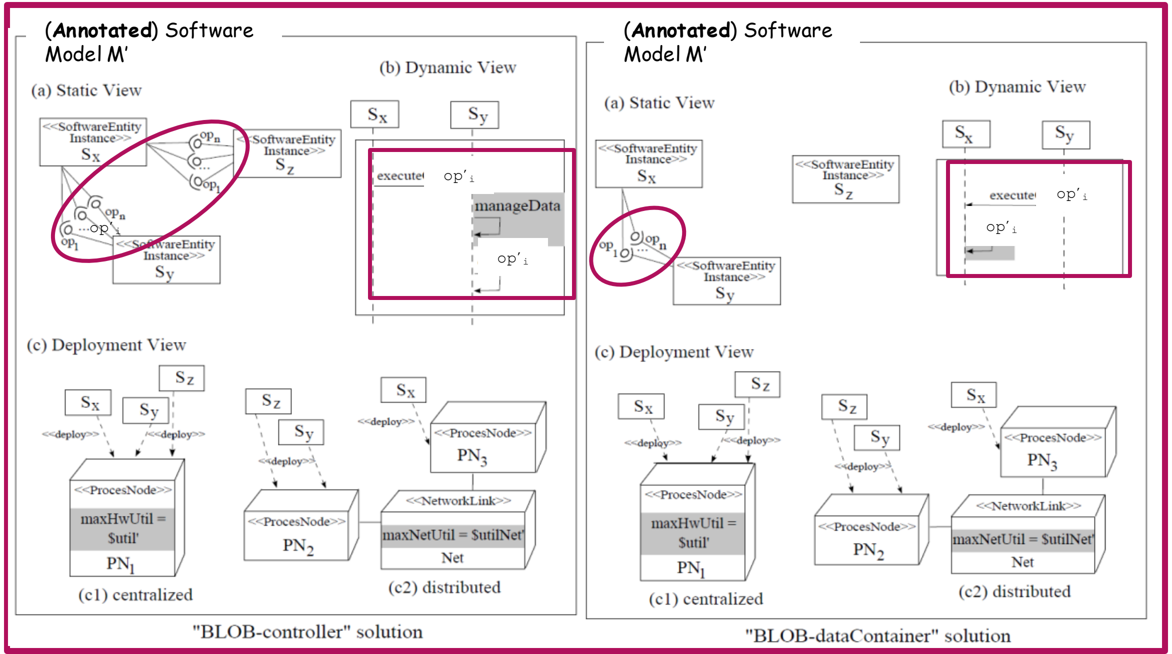
BLOB



Performance Antipatterns Representation

<u>SOLUTION</u>: *"Refactor the design to distribute intelligence uniformly over the applications top-level classes, and to keep related data and behavior together"*

BLOB



V. Cortellessa, A. Di Marco, and C. Trubiani.

"An approach for modeling and detecting Software Performance Antipatterns based on first-order logics",

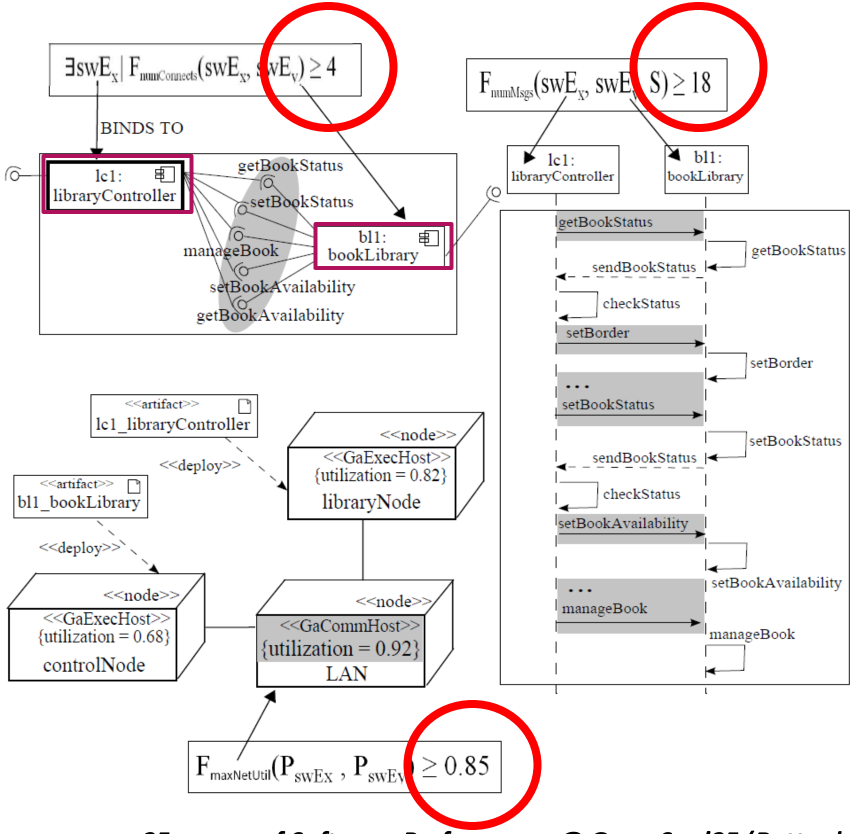
Software and Systems Modeling (SoSyM), 2014.

Performance Antipatterns Detection

thresholds

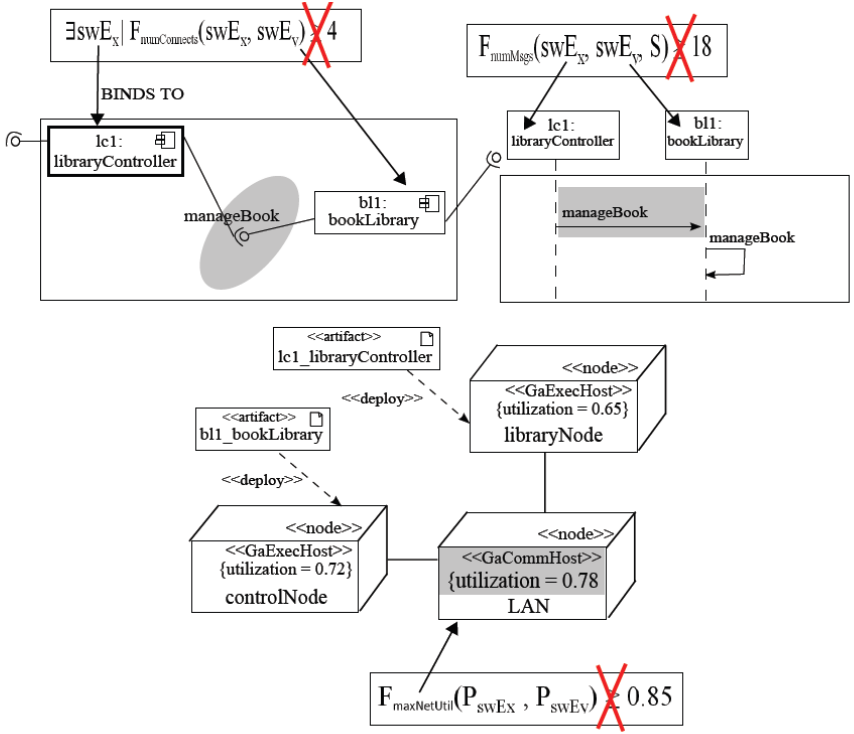
(libraryController, bookLibrary, browseCatalog)

This instance satisfies the **Blob** predicate, hence it must be pointed out to the designer for a deeper analysis



Performance Antipatterns Solution

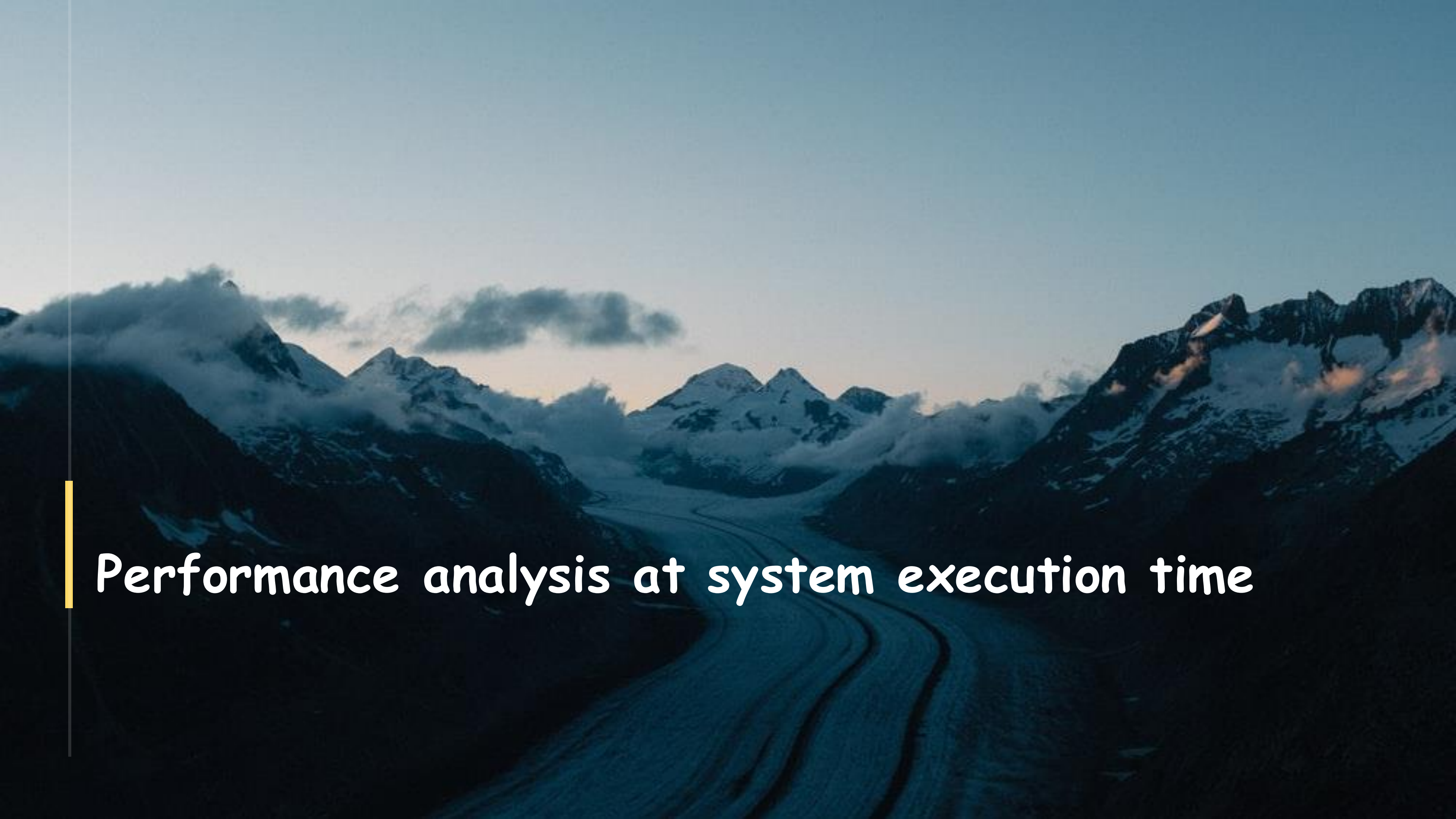
An example: solving a Blob instance



Solutions of an antipattern (i.e., **refactoring actions**) can be automatically deducted by negating detection predicates

… any relationship to sustainability?

- Separation of **software** and **hardware** for sustainability **modeling**
- Modeling languages for **representing sustainability metrics**
- **Refactoring impact** on sustainability
- Sustainability **antipatterns**



Performance analysis at system execution time

Some relevant literature (from the last 25+ years) (model reconstruction, diagnostic, root cause analysis)

- Tauseef A. Israr, Danny H. Lau, Greg Franks, C. Murray Woodside (2005):
  *Automatic generation of layered queuing **software performance models from commonly available traces***
- Ahmad Mizan, Greg Franks (2011):
  *An automatic **trace based performance evaluation model building** for parallel distributed systems*

- Thanh H. D. Nguyen, Bram Adams, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed N. Nasser, Parminder Flora (2012):
  *Automated **detection of performance regressions** using statistical process control techniques*
- Christoph Heger, Jens Happe, Roozbeh Farahbod (2013):
  *Automated **root cause isolation** of performance regressions during software development*
- David Daly, William Brown, Henrik Ingo, Jim O'Leary, David Bradford (2020):
  *The Use of Change Point Detection to **Identify Software Performance Regressions** in a Continuous Integration System*
- Raghu Ramakrishnan, Arvinder Kaur (2017):
  *Technique for Detecting **Early-Warning Signals of Performance Deterioration** in Large Scale Software Systems*
- Vittorio Cortellessa, Luca Traini (2020):
  *Detecting **Latency Degradation Patterns** in Service-based Systems*
- Yutong Zhao, Lu Xiao, Xiao Wang, Lei Sun, Bihuan Chen, Yang Liu, Andre B. Bondi (2020):
  *How Are **Performance Issues Caused and Resolved?-An Empirical Study** from a Design Perspective*

Our recent contributions

Time series forecasting

- *Federico Di Menna, Vittorio Cortellessa, Maurizio Lucianelli, Luca Sardo, Luca Traini:*
  RADig-X: a Tool for Regressions Analysis of User Digital Experience. *SANER 2024*
- *Federico Di Menna, Luca Traini, Vittorio Cortellessa:*
  Time Series Forecasting of Runtime Software Metrics: An Empirical Study. *ICPE 2024*
- *(((Using transformer models and metalearning – ongoing)))*

Warmup -vs- Steady state

- *Luca Traini, Vittorio Cortellessa, Daniele Di Pompeo, Michele Tucci:*
  Towards effective assessment of steady state performance in Java software: are we there yet? *EMSE journal (2023)*
- *Luca Traini, Federico Di Menna, Vittorio Cortellessa:*
  AI-driven Java Performance Testing: Balancing Result Quality with Testing Time. *ASE 2024*

Code analysis/optimization

- *Luca Traini, Daniele Di Pompeo, Michele Tucci, Bin Lin, Simone Scalabrino, Gabriele Bavota, Michele Lanza, Rocco Oliveto, Vittorio Cortellessa:*
  How Software Refactoring Impacts Execution Time. *ACM TOSEM (2022)*
- *F. Di Menna, L. Traini, G. Bavota, V. Cortellessa:*
  Investigating Execution-Aware Language Models for Code Optimization. *RENE@ICPC 2025*
- *(((Performance analysis of repositories that use AI libraries - ongoing)))*

Time Series Forecasting of Runtime Software Metrics: An Empirical Study

Federico Di Menna
federico.dimenna@graduate.univaq.it
University of L'Aquila
Italy

Luca Traini
luca.traini@univaq.it
University of L'Aquila
Italy

Vittorio Cortellessa
vittorio.cortellessa@univaq.it
University of L'Aquila
Italy

ABSTRACT

Software applications can produce a wide range of runtime software metrics (*e.g.*, number of crashes, response times), which can be closely monitored to ensure operational efficiency and prevent significant software failures. These metrics are typically recorded as time series data. However, runtime software monitoring has become a high-effort task due to the growing complexity of today's software systems. In this context, time series forecasting (TSF) offers unique opportunities to enhance software monitoring and facilitate proactive issue resolution. While TSF methods have been widely studied in areas like economics and weather forecasting, our understanding of their effectiveness for software runtime metrics remains somewhat limited.

In this paper, we investigate the effectiveness of four TSF methods on 25 real-world runtime software metrics recorded over a period of one and a half years. These methods comprise three recurrent neural network (RNN) models and one traditional time series analysis technique (i.e., SARIMA). The metrics are gathered from a large-scale IT infrastructure involving tens of thousands of digital devices. Our results indicate that, in general, RNN models are very effective in the runtime software metrics prediction, although in some scenarios and for certain specific metrics (*e.g.*, waiting times) SARIMA proves to outperform RNN models. Additionally, our findings suggest that the advantages of using RNN models vanish when the prediction horizon becomes too wide, in our case when it exceeds one week.

CCS CONCEPTS

(ICPE '24), May 7–11, 2024, London, United Kingdom. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3629526.3645049

1 INTRODUCTION

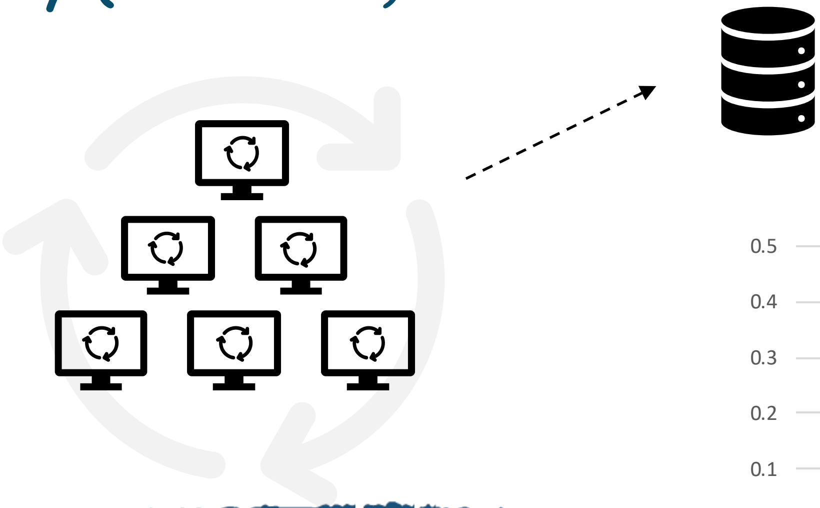
As software systems grow in complexity, the task of ensuring software quality becomes increasingly challenging. Today software systems constantly evolve, with frequent daily software releases [46], and they operate under highly variable workloads [5], which make them susceptible to unforeseen software failures [5, 55, 58]. In such a dynamic environment, traditional proactive strategies, such as software testing, are often insufficient for ensuring consistent operational efficiency [45, 58]. For this reason, monitoring is emerging as a key activity for maintaining operational efficiency of software systems [12, 23, 32].

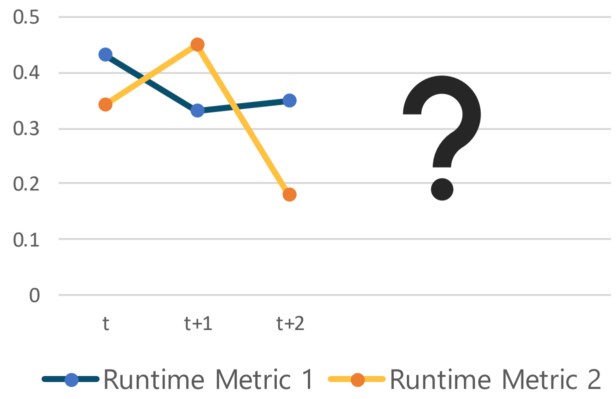
Modern software applications can produce large volumes of runtime metrics, which are typically stored as time series data in specialized databases [23], (*e.g.*, Prometheus [15]). Dedicated monitoring teams continuously analyze these time series to identify and mitigate potential software issues [12]. However, the vast volume of collected data can make manual analysis costly and potentially ineffective. To address this challenge, researchers started to develop automated techniques that can facilitate the identification of software issues or aid in the debugging process [1, 6, 16, 19, 25, 53].

Despite these advancements, significant opportunities in the realm of data analysis remain unexploited. Time series forecasting (TSF), in particular, presents a promising avenue for enhancing the current practices in software monitoring. Indeed, the ability to predict future trends in runtime software metrics could facilitate

F. Di Menna, V. Cortellessa, L. Traini

Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

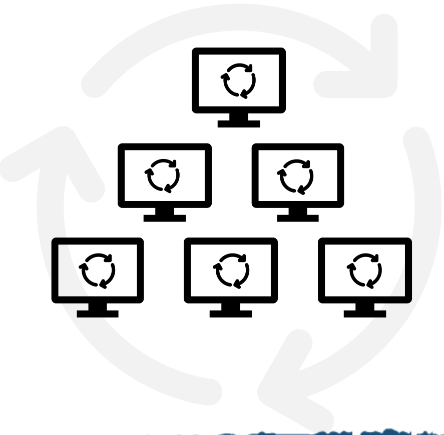


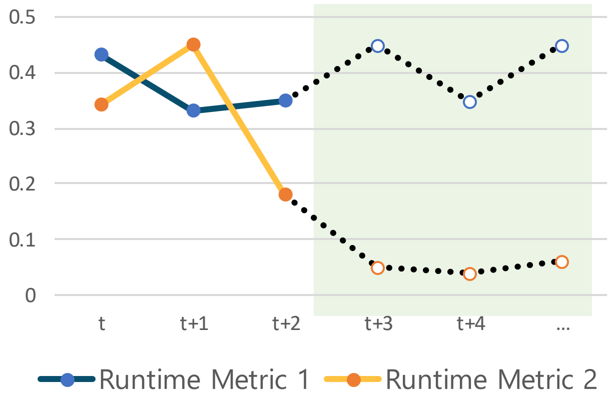


Modern software systems are continuously monitored to manage uncertainty during their evolution

F. Di Menna, V. Cortellessa, L. Traini

Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)



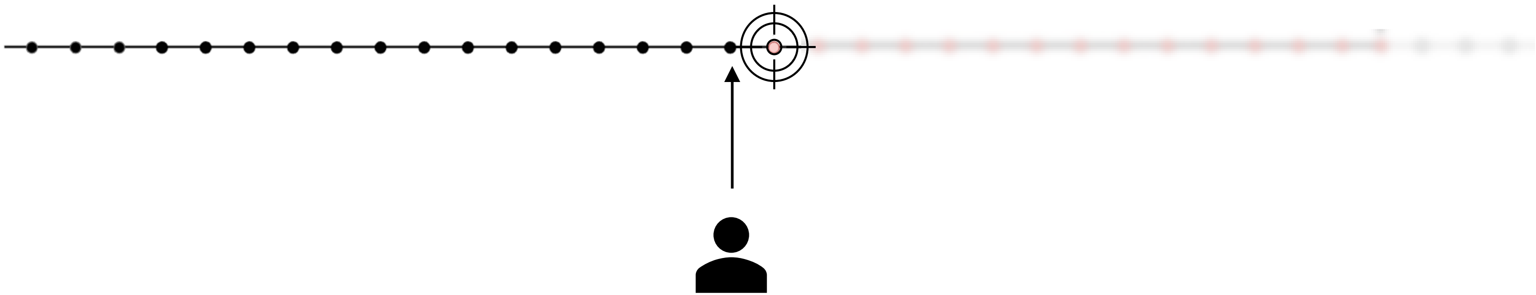


Monitored data can be used to forecast the (runtime) behavior of software systems

F. Di Menna, V. Cortellessa, L. Traini

Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

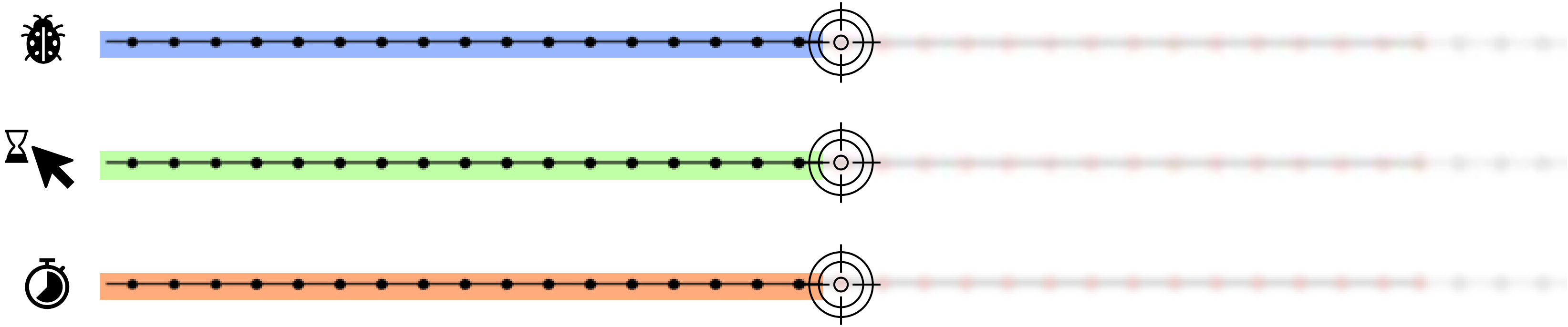
- Exploring in detail the effectiveness of TSF approaches when applied to runtime software metrics



How effective are TSF methods when applied to predict short-term runtime software metrics?

F. Di Menna, V. Cortellessa, L. Traini
Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

- Exploring in detail the effectiveness of TSF approaches when applied to runtime software metrics

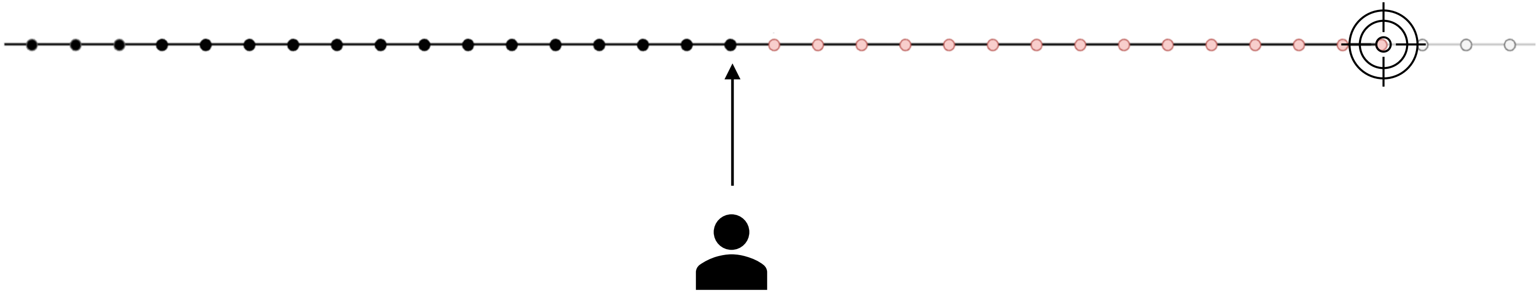


Do TSF methods exhibit diverse forecasting accuracy over different classes of runtime software metrics?

F. Di Menna, V. Cortellessa, L. Traini

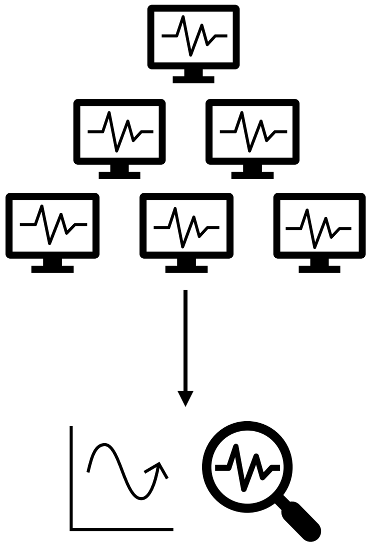
Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

- Exploring in detail the effectiveness of TSF approaches when applied to runtime software metrics



To what extent does forecasting accuracy degrade when applied to predict longer-term runtime software metrics?

F. Di Menna, V. Cortellessa, L. Traini
Time Series Forecasting of Runtime Software Metrics: An Empirical Study *(ICPE 2024)*



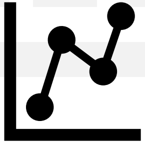
- **Tens of thousands of monitored digital devices for more than one year**
- Total of **25 runtime software metrics**
- Three software metrics classes:
  - **Crash rates (9)** : Average number of observed crashes of an application per hour
  - **Hang times (8)** : Percentage of idle time of an application
  - **Waiting times (8)** : User response time

F. Di Menna, V. Cortellessa, L. Traini

Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

Statistical

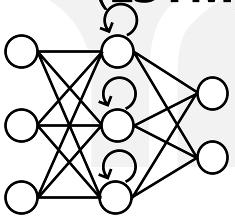
Seasonal Auto-Regressive Integrated Moving Average (**SARIMA**)



Machine Learning

Fully-Connected Recurrent Neural Network (**FC-RNN**)

Long Short-Term Memory (**LSTM**)



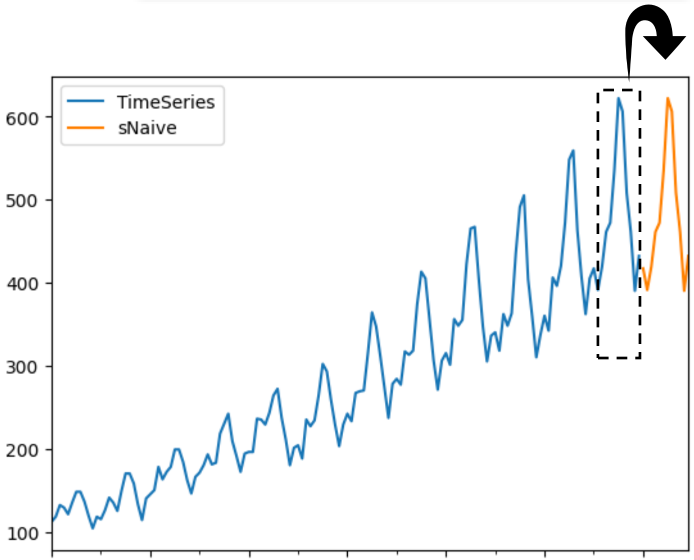
Gated Recurrent Unit (**GRU**)

F. Di Menna, V. Cortellessa, L. Traini

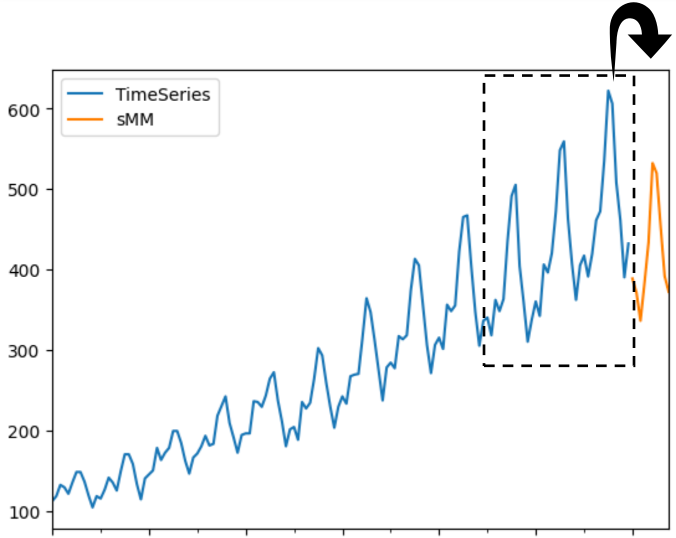
Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

- Two naïve baseline approaches

Seasonal Naïve (sNaïve)



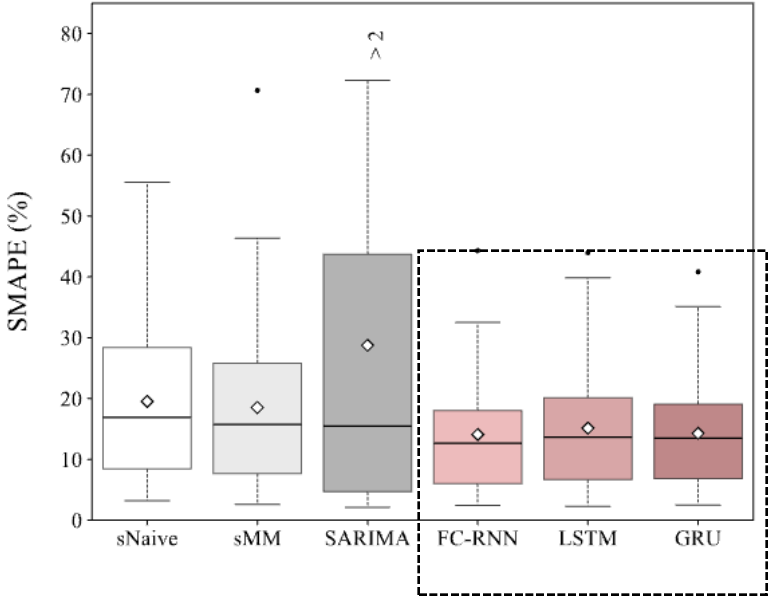
Seasonal Monthly Mean (sMM)



F. Di Menna, V. Cortellessa, L. Traini

Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

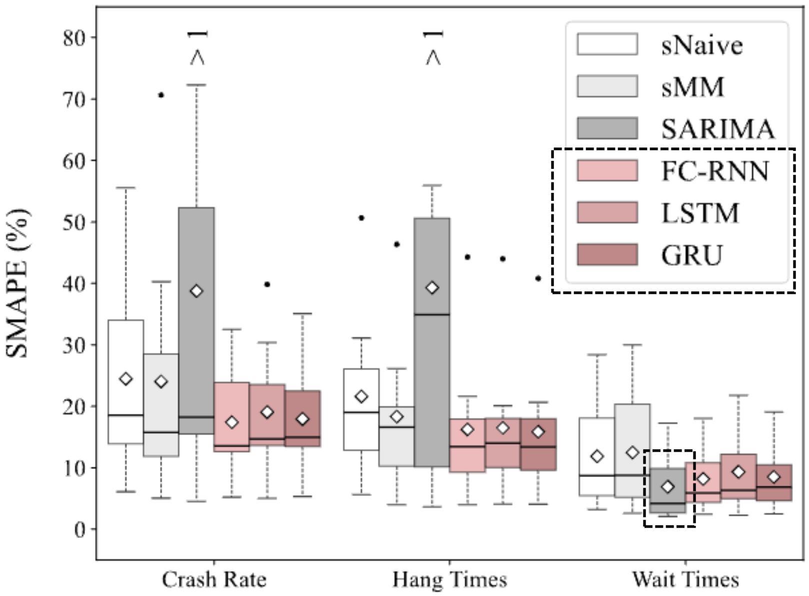
How effective are TSF methods when applied to predict <u>short-term</u> runtime software metrics?



F. Di Menna, V. Cortellessa, L. Traini

Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

Do TSF methods exhibit diverse forecasting accuracy over different classes of runtime software metrics?

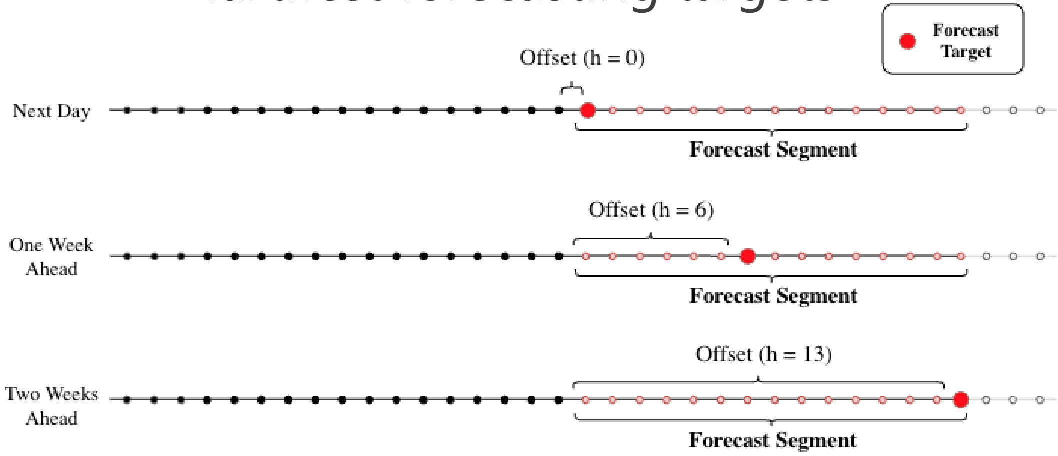


F. Di Menna, V. Cortellessa, L. Traini

Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

To what extent does forecasting accuracy degrade when applied to predict <u>longer-term</u> runtime software metrics?

Exploitation of the **prediction offset** to estimate farthest forecasting targets

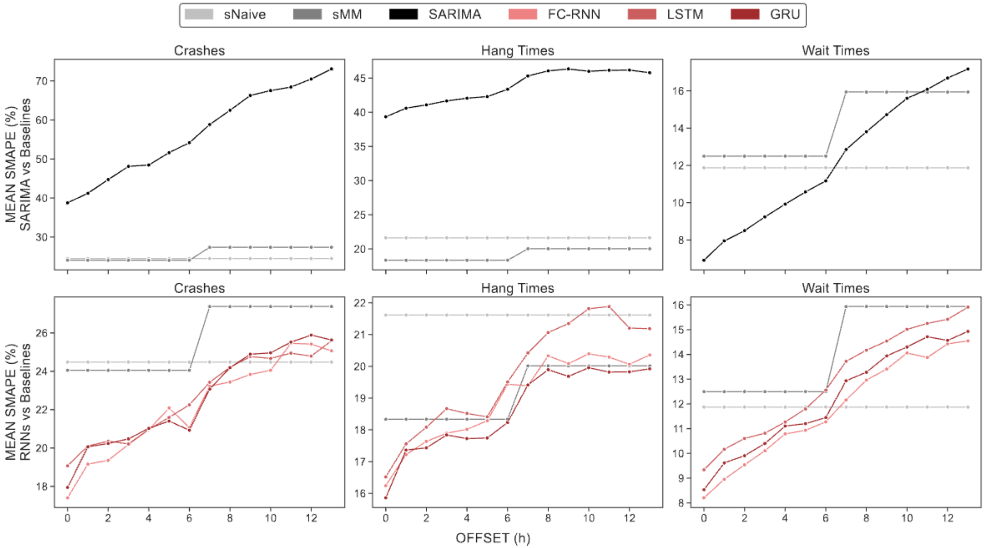


F. Di Menna, V. Cortellessa, L. Traini

Time Series Forecasting of Runtime Software Metrics: An Empirical Study (*ICPE 2024*)

To what extent does forecasting accuracy degrade when applied to predict <u>longer-term</u> runtime software metrics?

- SMAPE values demonstrate a rising trend as the offset increases
- Effectiveness of TSF methods vanish if the offset exceeds approximately one week



… any relationship to sustainability?

- Separation of **software** and **hardware** for sustainability **modeling**
- Modeling languages for **representing sustainability metrics**
- **Refactoring impact** on sustainability
- Sustainability **antipatterns**
- Sustainability **metrics time series forecasting**

2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)

AI-driven Java Performance Testing: Balancing Result Quality with Testing Time

Luca Traini
luca.traini@univaq.it
University of L'Aquila, Italy

Federico Di Menna
federico.dimenna@graduate.univaq.it
University of L'Aquila, Italy

Vittorio Cortellessa
vittorio.cortellessa@univaq.it
University of L'Aquila, Italy

ABSTRACT

Performance testing aims at uncovering efficiency issues of software systems. In order to be both effective and practical, the design of a performance test must achieve a reasonable trade-off between result quality and testing time. This becomes particularly challenging in Java context, where the software undergoes a warm-up phase of execution, due to just-in-time compilation. During this phase, performance measurements are subject to severe fluctuations, which may adversely affect quality of performance test results. Both practitioners and researchers have proposed approaches to mitigate this issue. Practitioners typically rely on a fixed number of iterated executions that are used to warm-up the software before starting to collect performance measurements (*state-of-practice*). Researchers have developed techniques that can dynamically stop warm-up iterations at runtime (*state-of-the-art*). However, these approaches often provide suboptimal estimates of the warm-up phase, resulting in either insufficient or excessive warm-up iterations, which may degrade result quality or increase testing time. There is still a lack of consensus on how to properly address this problem. Here, we propose and study an AI-based framework to dynamically halt warm-up iterations at runtime. Specifically, our framework leverages recent advances in AI for Time Series Classification (TSC) to predict the end of the warm-up phase during test execution. We conduct experiments by training three different TSC models on half a million of measurement segments obtained from JMH microbenchmark executions. We find that our framework significantly improves the accuracy of the warm-up estimates provided by *state-of-practice* and *state-of-the-art* methods. This higher estimation accuracy results in a net improvement in either result quality or testing time for up to +35.3% of the microbenchmarks. Our study highlights that integrating AI to dynamically estimate the end of the warm-up phase can enhance the cost-effectiveness of Java performance testing.

ACM Reference Format:
Luca Traini, Federico Di Menna, and Vittorio Cortellessa. 2024. AI-driven Java Performance Testing: Balancing Result Quality with Testing Time. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3691620.3695017

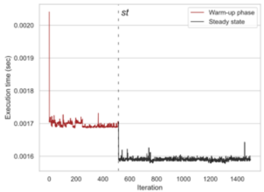


Figure 1: Execution time of a Java microbenchmark (from the *Roaring Bitmap* project) over consecutive iterations. The execution is characterized by an initial warm-up phase and a subsequent steady-state of performance. The grey dotted line indicates the iteration *st* at which steady-state is attained.

1 INTRODUCTION

Software performance is a critical non-functional aspect of software systems. Technology organizations use performance testing to uncover performance bugs that might deteriorate software efficiency. Nevertheless, performance testing requires careful design in order to be effective. A significant challenge is to design an adequate num-

L. Traini, F. Di Menna, V. Cortellessa
AI-driven Java Performance Testing: Balancing Result Quality with Testing Time (*ASE 2024*)

Performance testing requires careful design in order to be effective

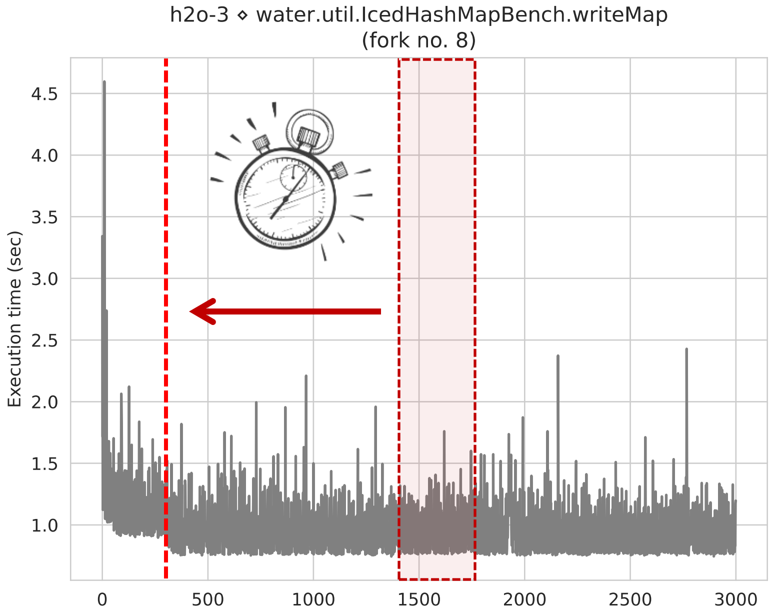
- Adequate number of execution repetitions that mitigate the **variability** of performance measurements

- Balance the quality of performance test results against the practical constraints of resources and **testing time**

L. Traini, F. Di Menna, V. Cortellessa
AI-driven Java Performance Testing: Balancing Result Quality with Testing Time (*ASE 2024*)

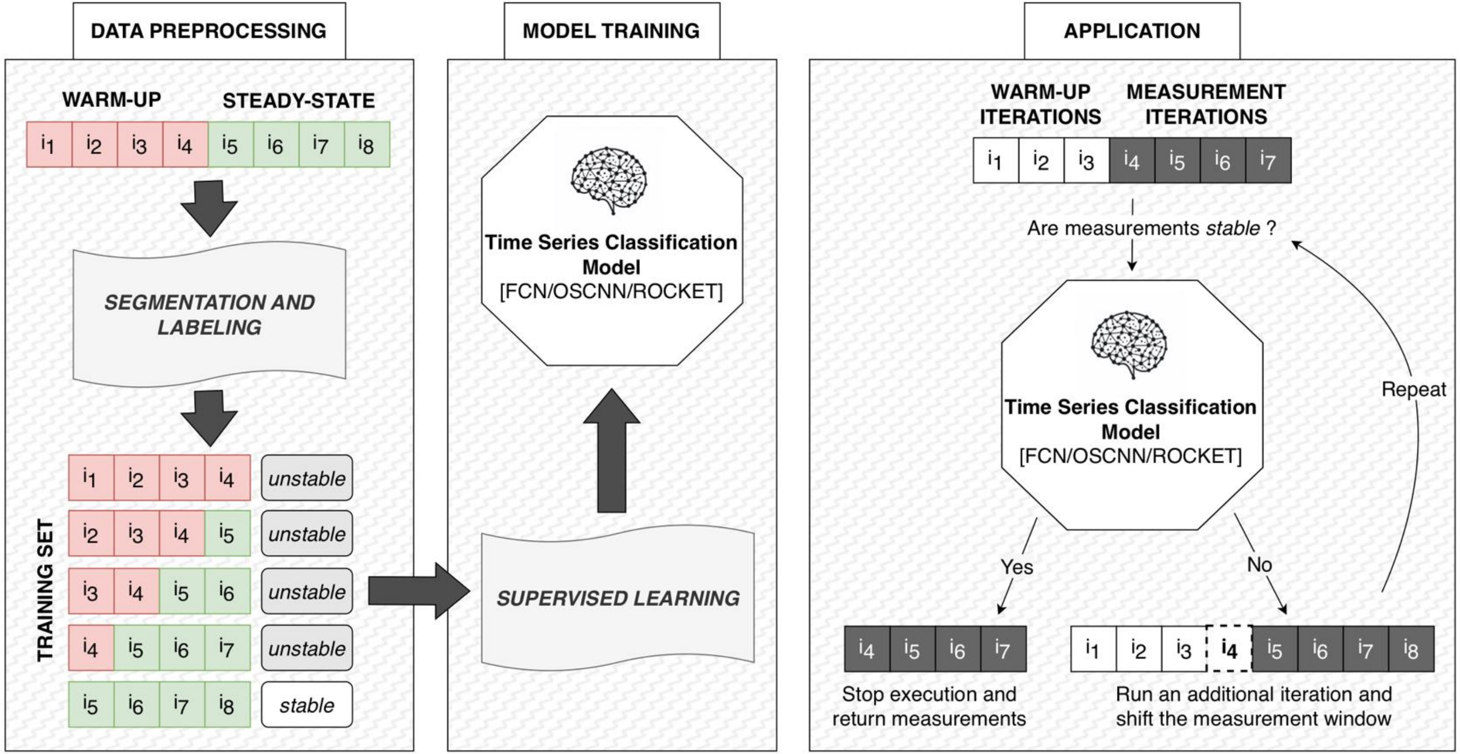
JVM - the warmup problem

```
long measurements[] = new long[3000];

for (int i = 0; i < 3000; i++) {
        long startTime = System.nanoTime();
        Arrays.sort(arr);
        long endTime = System.nanoTime();
        long duration = (endTime - startTime);
        measurements[i] = duration;

}
```



L. Traini, F. Di Menna, V. Cortellessa

AI-driven Java Performance Testing: Balancing Result Quality with Testing Time (*ASE 2024*)



L. Traini, F. Di Menna, V. Cortellessa
AI-driven Java Performance Testing: Balancing Result Quality with Testing Time (*ASE 2024*)

- **586** JMH microbenchmarks across **30** Java software system
- **5K+** time series, each one involving measures from **3,000** consecutive microbenchmark iterations

Final dataset of **521,900 measurement segments:**

- 376,925 (72%) stable segments
- 144,975 (28%) unstable segments

Repository	Stars	Forked Repo.	Microbench.
HdrHistogram/HdrHistogram	2,141	251	20
JCTools/JCTools	3,496	554	20
ReactiveX/RxJava	47,702	7,581	20
RoaringBitmap/RoaringBitmap	3,415	535	20
apache/arrow	13,686	3,341	20
apache/camel	5,366	4,896	20
apache/hive	5,370	4,601	20
apache/kafka	27,594	13,571	20
apache/logging-log4j2	3,290	1,559	20
apache/tinkerpop	1,911	786	20
cantaloupe-project/cantaloupe	261	104	19
crate/crate	3,977	546	20
eclipse-vertx/vert.x	14,153	2,043	16
eclipse/eclipse-collections	2,368	581	20
eclipse/jetty.project	3,766	1,899	19
eclipse/rdf4j	347	160	20
h2oai/h2o-3	6,756	1,991	20
hazelcast/hazelcast	5,935	1,803	17
imglib/imglib2	291	93	20
jdbi/jdbi	1,917	333	15
jgrapht/jgrapht	2,535	819	20
netty/netty	32,923	15,763	20
openzipkin/zipkin	16,780	3,069	20
prestodb/presto	15,646	5,265	20
prometheus/client_java	2,134	772	20
protostuff/protostuff	2,016	302	20
r2dbc/r2dbc-h2	196	44	20
raphw/byte-buddy	6,052	776	20
yellowstonegames/SquidLib	447	46	20
zalando/logbook	1,737	257	20

L. Traini, F. Di Menna, V. Cortellessa
AI-driven Java Performance Testing: Balancing Result Quality with Testing Time (*ASE 2024*)

Model	Prec.	Rec.	F1	Bal. Acc.
FCN	0.880	0.659	0.753	0.712
OSCNN	0.886	0.650	0.748	0.715
ROCKET	0.810	0.932	0.867	0.682

- **Balanced Accuracy** is the average recall obtained across each of the two classes. We use this metric instead of traditional accuracy due to the imbalanced nature of our dataset.

TSC models demonstrated their **suitability** for dynamically halting warm-up iterations. This supports their integration into our framework.

AI-based framework vs. state-of-practice (SOP)

Model *vs.* SOP	Improvement (%)			Regression (%)			Net Improvement (%)
	Res. Quality	Testing Time	*Total*	Res. Quality	Testing Time	*Total*	(*Tot. Impr. - Tot. Regr.*)
FCN *vs.* SOP	16.7	30.7	***47.4***	14.3	7.8	***22.2***	*+25.3*
OSCNN *vs.* SOP	17.9	30.9	***48.8***	13.7	8.2	***21.8***	*+27.0*
Rocket *vs.* SOP	9.4	20.1	***29.5***	25.9	6.5	***32.4***	*-2.9*

AI-based framework provides more accurate estimates of the warm-up phase compared to the SOP.
Net improvement in either result quality or testing time **in up to +27%** of the microbenchmarks,
with OSCNN demonstrating the highest net improvement.

AI-based framework vs. state-of-the-art (SOTA)

Model *vs.* SOTA	**Improvement (%)** Res. Quality	Testing Time	***Total***	**Regression (%)** Res. Quality	Testing Time	***Total***	**Net Improvement (%)** (*Tot. Impr. - Tot. Regr.*)
FCN *vs.* CV	26.8	27.1	***53.9***	12.3	7.7	***20.0***	***+34.0***
FCN *vs.* RCIW	6.3	50.3	***56.7***	24.2	4.6	***28.8***	***+27.8***
FCN *vs.* KLD	25.9	24.4	***50.3***	13.1	10.4	***23.5***	***+26.8***
OSCNN *vs.* CV	28.8	26.8	***55.6***	12.3	8.0	***20.3***	***+35.3***
OSCNN *vs.* RCIW	7.0	52.6	***59.6***	21.8	4.8	***26.6***	***+32.9***
OSCNN *vs.* KLD	27.8	23.0	***50.9***	12.1	12.8	***24.9***	***+25.9***
ROCKET *vs.* CV	11.3	19.8	***31.1***	24.6	2.7	***27.3***	***+3.8***
ROCKET *vs.* RCIW	4.1	24.4	***28.5***	51.4	3.4	***54.8***	***-26.3***
ROCKET *vs.* KLD	7.2	21.5	***28.7***	22.5	3.9	***26.5***	***+2.2***

AI-based framework provides more accurate estimates of the warm-up phase than the SOTA.
Variants of the framework based on **neural network** models observably enhance
either the result quality or testing time of the SOTA techniques,
leading to net improvements in **up to +35.3%** of the microbenchmarks.

… any relationship to sustainability?

- Separation of **software** and **hardware** for sustainability **modeling**
- Modeling languages for **representing sustainability metrics**
- **Refactoring impact** on sustainability
- Sustainability **antipatterns**
- Sustainability **metrics time series forecasting**
- **Warmup vs Steady state** analysis for sustainability metrics

Investigating Execution-Aware Language Models for Code Optimization

Federico Di Menna[†], Luca Traini[†], Gabriele Bavota[‡], Vittorio Cortellessa[†]
[†]*University of L'Aquila, L'Aquila, Italy*
[‡]*Software Institute - Università della Svizzera Italiana, Switzerland*
federico.dimenna@graduate.univaq.it, luca.traini@univaq.it, gabriele.bavota@usi.ch, vittorio.cortellessa@univaq.it

***Abstract*—Code optimization is the process of enhancing code efficiency, while preserving its intended functionality. This process often requires a deep understanding of the code execution behavior at run-time to identify and address inefficiencies effectively. Recent studies have shown that language models can play a significant role in automating code optimization. However, these models may have insufficient knowledge of how code execute at run-time. To address this limitation, researchers have developed strategies that integrate code execution information into language models. These strategies have shown promise, enhancing the effectiveness of language models in various software engineering tasks. However, despite the close relationship between code execution behavior and efficiency, the specific impact of these strategies on code optimization remains largely unexplored. This study investigates how incorporating code execution information into language models affects their ability to optimize code. Specifically, we apply three different training strategies to incorporate four code execution aspects — line executions, line coverage, branch coverage, and variable states — into CodeT5+, a well-known language model for code. Our results indicate that execution-aware models provide limited benefits compared to the standard CodeT5+ model in optimizing code.**

***Index Terms*—Code Optimization, Deep Learning**

I. INTRODUCTION

have shown that integrating these models with code execution information can significantly enhance their effectiveness across a variety of downstream software engineering tasks [15]–[18]. For instance, Ding *et al.* [18] proposed a pre-training strategy to teach language models specific aspects of code execution, such as branch coverage and variable states, demonstrating improvements in tasks like clone retrieval and vulnerability detection. Similarly, Ni *et al.* [15] introduced *NExT*, a method that enables language models to inspect variable states of executed code lines and reason about their execution behavior, resulting in a higher fix rate for program repair tasks. Despite these and other efforts [16], [17], [19]–[21], the impact of execution-awareness in automated code optimization remains largely unexplored.

Given the close relationship between run-time execution behavior and code efficiency, this paper investigates how teaching language models to understand code execution behavior affects their effectiveness in optimizing code. Specifically, we first train a CodeT5+ model [22] with training objectives related to four code execution aspects, namely number of line executions, line coverage, branch coverage, and variable states.

F. Di Menna, L. Traini, G. Bavota, V. Cortellessa
Investigating Execution-Aware Language Models for Code Optimization (RENE@ICPC 2025)

Deep-Learning (especially Large Language Models) engaged on programming language modeling for advancing **code intelligence**:

- Code Generation and Summarization
- Code Completion
- Code Review and Bug Detection
- **Code Optimization**
- …

SUPERSONIC: Learning to Generate Source Code Optimizations in C/C++

Zimin Chen, Sen Fang, and Martin Monperrus

DeepDev-PERF: A Deep Learning-Based Approach for Improving Software Performance

Spandan Garg, Roshanak Zilouchian Moghaddam, Colin B. Clement

Published as a conference paper at ICLR 2024

LEARNING PERFORMANCE-IMPROVING CODE EDITS

Alexander Shypula[1], Aman Madaan[2], Yimeng Zeng[1], Uri Alon[4], Jacob Gardner[1], Milad Hashemi[3], Graham Neubig[2], Parthasarathy Ranganathan[3], Osbert Bastani[1], Amir Yazdanbakhsh[4]

[1]University of Pennsylvania, [2]Carnegie Mellon University, [3]Google [4]Google DeepMind
shypula@seas.upenn.edu and ayazdan@google.com

ABSTRACT

With the waning of Moore's law, optimizing program performance has become a major focus of software research. However, high-level optimizations such as API and algorithm changes remain elusive due to the difficulty of understanding the semantics of code. Simultaneously, pretrained large language models (LLMs) have demonstrated strong capabilities at solving a wide range of programming tasks. To that end, we introduce a framework for adapting LLMs to high-level program optimization. First, we curate a dataset of performance-improving edits made by human programmers of over 77 K competitive C++ programming submission pairs, accompanied by extensive unit tests. A major challenge is the significant variability of measuring performance on commodity hardware, which can lead to spurious "improvements". To isolate and reliably evaluate the impact of program optimizations, we design an environment based on the gem5 full system simulator, the de facto simulator used in academia and industry. Next, we propose a broad range of adaptation strategies for code optimization; for prompting, these include retrieval-based few-shot prompting and chain-of-thought, and for finetuning, these include performance-conditioned generation and synthetic data augmentation based on self-play. A combination of these techniques achieves a mean speedup of 6.86× with eight generations, higher than average optimizations from individual programmers (3.66×). Using our model's fastest generations, we set a new upper limit on the fastest speedup possible for our dataset at 9.64× compared to using the fastest human submissions available (9.56×).[1]

1 INTRODUCTION

Despite the impressive progress of optimizing compilers and other tools for performance engineering (Aho et al., 2007), programmers are still largely responsible for high-level performance considerations such as algorithms and API choices. Recent work has demonstrated the promise of deep learning for automating performance optimization (Garg et al., 2022; Mankowitz et al., 2023). However, these techniques are either narrow or difficult to build on due to the lack of open datasets and lack of reliable performance measurement techniques, which has stymied research in this direction. Recently, pre-trained large language models (LLMs) have demonstrated impressive performance at a wide range of programming tasks (Chen et al., 2021b; Fried et al., 2022; Xu et al., 2022; Nijkamp et al., 2022). Yet, the effectiveness of large, pre-trained LLMs for program optimization remains an open research question. We study whether such LLMs can be

F. Di Menna, L. Traini, G. Bavota, V. Cortellessa
Investigating Execution-Aware Language Models for Code Optimization (RENE@ICPC 2025)

Code representation learning is a **key factor** for LLMs effectiveness:

- **Dynamic (execution) information** is critical in code understanding
- Recent work demonstrated **benefits** leveraging execution-related information while performing code-related tasks with LLMs
- **The task of code optimization is closely linked to both code understanding and execution behavior**

Goal: exploit execution-aware information for improving the code optimization task


CODE REPRESENTATION PRE-TRAINING WITH COMPLEMENTS FROM PROGRAM EXECUTIONS

Jiabo Huang, Jianyu Zhao, Yuyang Rong, Yiwen Guo, Yifeng He, Hao Chen

LExecutor: Learning-Guided Execution

Beatriz Souza, University of Stuttgart, Stuttgart, Germany

Michael Pradel, University of Stuttgart, Stuttgart, Germany

2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)

TRACED: Execution-aware Pre-training for Source Code

Yangruibo Ding, Columbia University, New York, NY, USA

Benjamin Steenhoek, Iowa State University, Ames, IA, USA

Kexin Pei, Columbia University, New York, NY, USA

Gail Kaiser, Columbia University, New York, NY, USA

Wei Le, Iowa State University, Ames, IA, USA

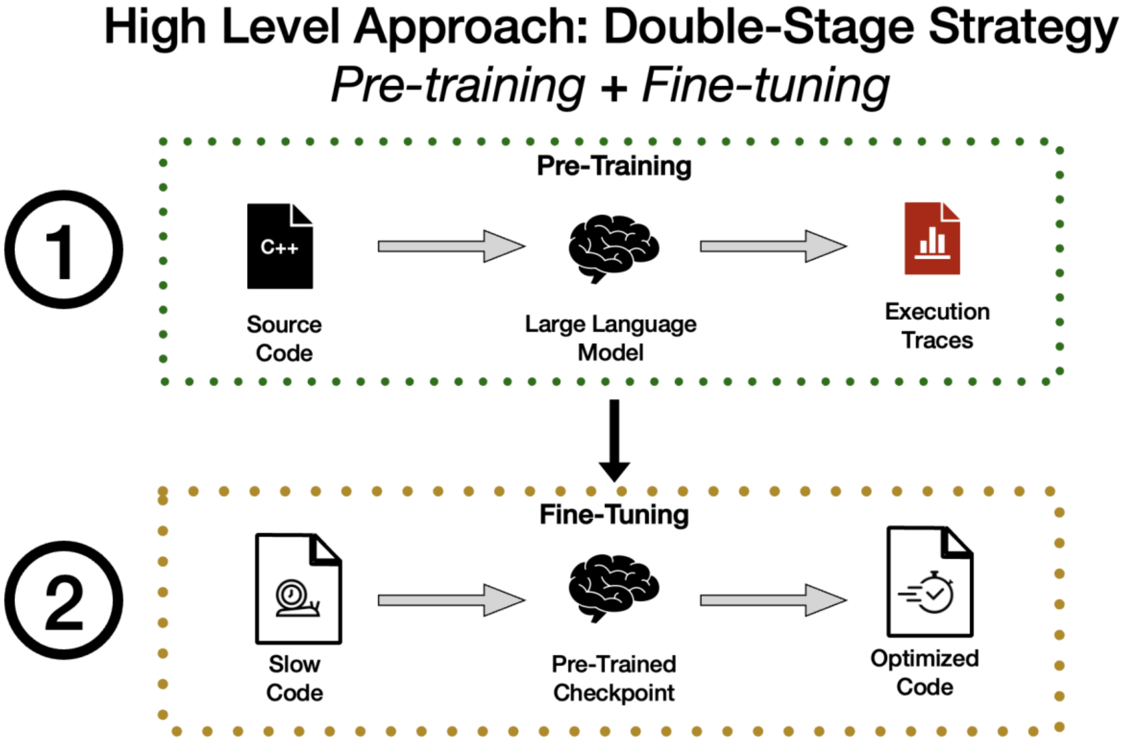
Baishakhi Ray, Columbia University, New York, NY, USA

ABSTRACT

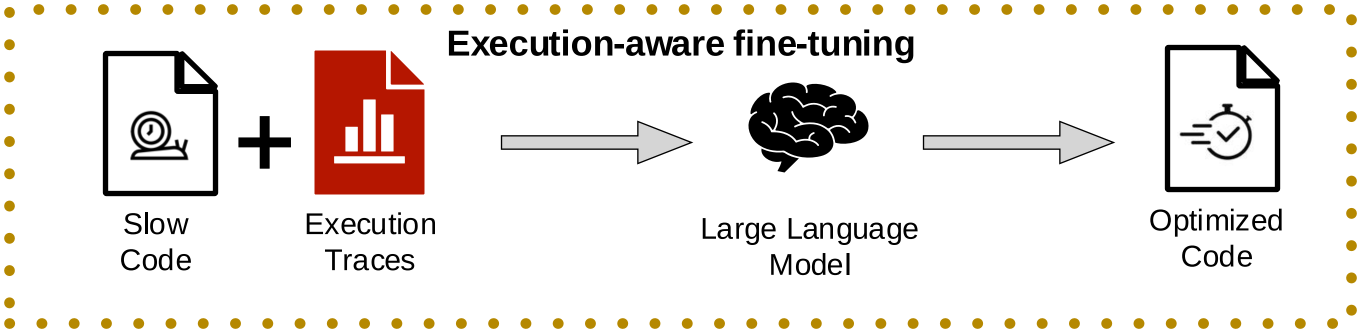
1 INTRODUCTION

F. Di Menna, L. Traini, G. Bavota, V. Cortellessa

Investigating Execution-Aware Language Models for Code Optimization (RENE@ICPC 2025)



High Level Approach: Direct Fine-Tuning Variant



Candidates for Experimental Evaluation

4 **execution-aware** aspects:

- Line Executions
- Line Coverage
- Branch Coverage
- Final Program States

✘

3 **training strategies**:

- S1 — Execution Prediction & Code Optimization
- S2 — Execution Prediction-MLM & Code Optimization
- S3 — Execution-Aware Code Optimization

VS

Baseline

Vanilla Model: Directly fine-tuned for Code Optimization

F. Di Menna, L. Traini, G. Bavota, V. Cortellessa
Investigating Execution-Aware Language Models for Code Optimization *(RENE@ICPC 2025)*

Model	Execution Aspect	Training Strategy		Evaluation Metrics		
		Pre-training	**Fine-tuning**	**Correct**	**Speedup**	**%Opt**
$BL_{S_{12}}$	-	-	Code optimization	**18.75%**	**1.79**	**7.68%**
LE_{S_1}	*Line Executions*	Execution-aware	Code optimization	12.36%	1.52	5.73%
LE_{S_2}	*Line Executions*	Execution-aware + MLM	Code optimization	11.97%	1.54	5.6%
LC_{S_1}	*Line Coverage*	Execution-aware	Code optimization	14.84%	1.7	6.9%
LC_{S_2}	*Line Coverage*	Execution-aware + MLM	Code optimization	14.45%	1.76	7.55%
BC_{S_1}	*Branch Coverage*	Execution-aware	Code optimization	13.93%	1.67	7.03%
BC_{S_2}	*Branch Coverage*	Execution-aware + MLM	Code optimization	13.15%	1.55	5.73%
VS_{S_1}	*Variable States*	Execution-aware	Code optimization	15.49%	1.69	7.29%
VS_{S_2}	*Variable States*	Execution-aware + MLM	Code optimization	14.97%	1.65	6.64%
BL_{S_3}	-	-	Code optimization	12.06%	2.09	9.22%
LE_{S_3}	*Line Executions*	-	Execution-aware code optimization	**12.71%**	**2.11**	**9.35%**
LC_{S_3}	*Line Coverage*	-	Execution-aware code optimization	9.97%	1.67	5.84%
BC_{S_3}	*Branch Coverage*	-	Execution-aware code optimization	8.65%	1.64	5.76%
VS_{S_3}	*Variable States*	-	Execution-aware code optimization	11.55%	1.96	8.35%

… any relationship to sustainability?

- Separation of **software** and **hardware** for sustainability **modeling**
- Modeling languages for **representing sustainability metrics**
- **Refactoring impact** on sustainability
- Sustainability **antipatterns**
- Sustainability **metrics time series forecasting**
- **Warmup vs Steady state** analysis for sustainability metrics
- **Code optimization** for sustainability purposes

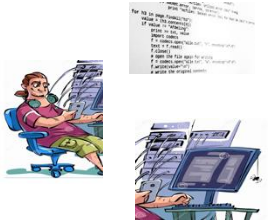
Conclusions

Other promising directions in SPE…

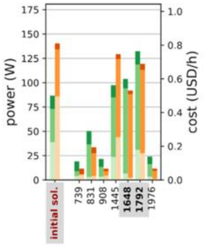
- **Performance in a CD/CI context (model+code)**
- Cross-fertilization of (black- and white-box) **models and runtime data**
- Automation **from functional testing to performance testing**
- **(Micro-)benchmarking** (and their settings/parameters)
- **Performance-driven developer’s assistant**
- How to profitably use **static code information**?

Model

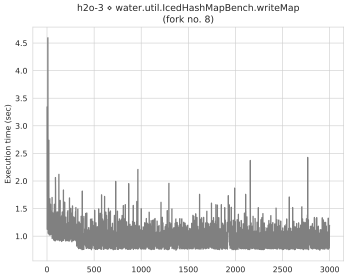
Code



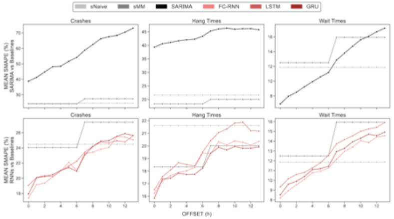
Separation of **software** and **hardware** for sustainability **modeling**



Refactoring impact on sustainability

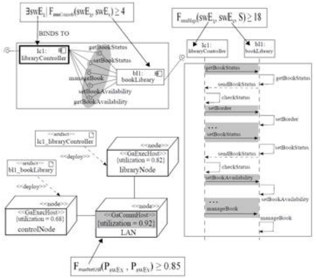


Warmup vs Steady state analysis for sustainability metrics

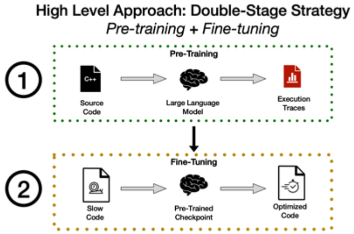


Sustainability **metrics time series forecasting**

Modeling languages for **representing sustainability metrics**



Sustainability **antipatterns**



Code optimization for sustainability purposes

Thanks!

Any question?