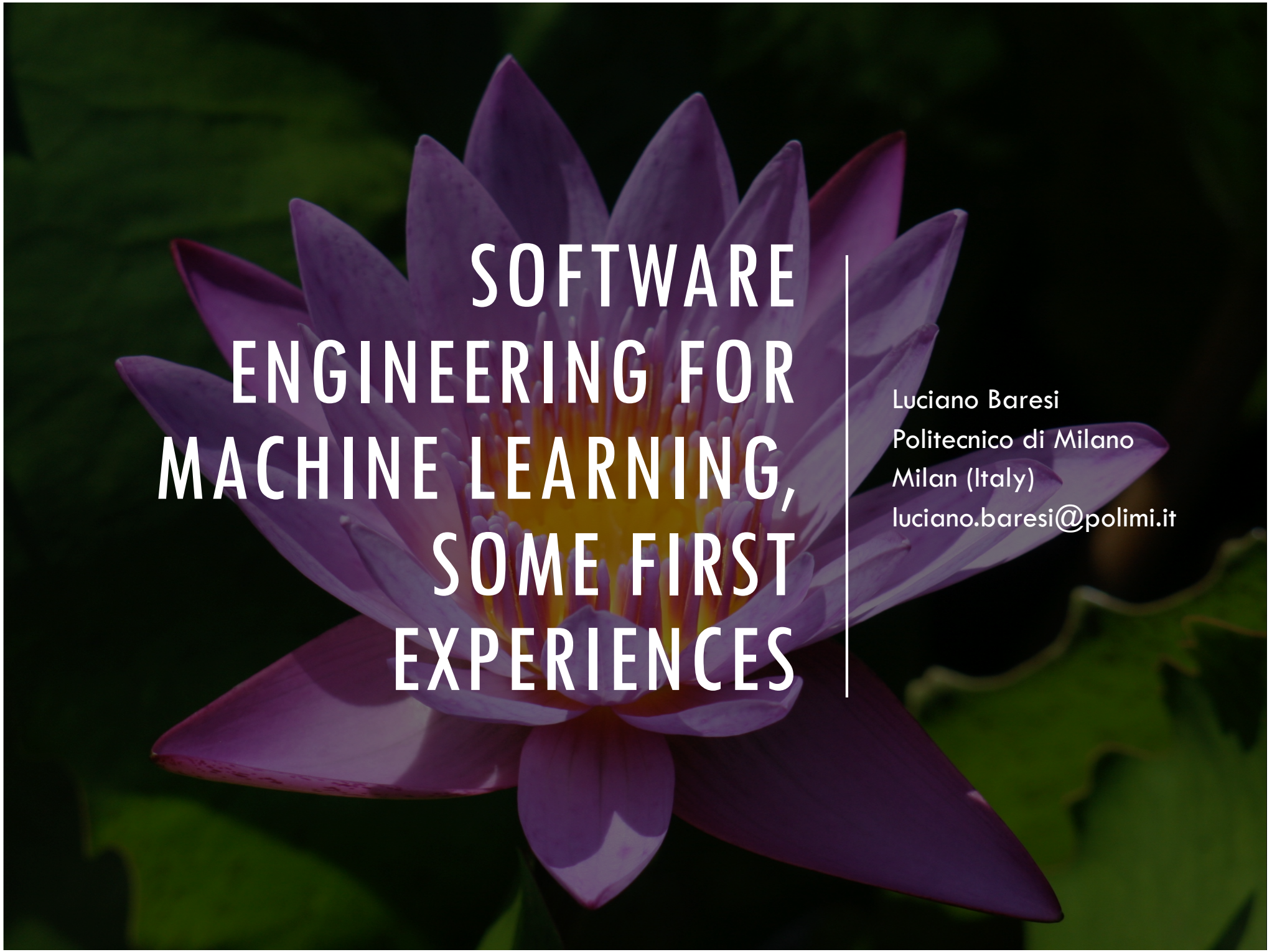


# SOFTWARE ENGINEERING FOR MACHINE LEARNING, SOME FIRST EXPERIENCES

Luciano Baresi
Politecnico di Milano
Milan (Italy)
luciano.baresi@polimi.it

# LUCIANO BARESI

- Professor @ DEIB
  - Visiting researcher/professor
    - University of Oregon (USA)
    - University of Paderborn (Germany)
    - Tongji University (China)
- Research interests
  - Software engineering
    - Dynamic software architectures
    - Software engineering for ML
    - Self-adaptive systems
    - Mobile applications

https://baresi.faculty.polimi.it

# CREDITS

Sam Guinea
Davide Yi Xian Hu
Giovanni Quattrocchi
Clément Quinton
Nicholas Rasi

Zoltán Ádám Mann
Andreas Metzger
Klaus Pohl


DISCLAIMER
Artificial Intelligence vs
Machine Learning

# DATA SCIENTIST VS SOFTWARE ENGINEER

Data scientist

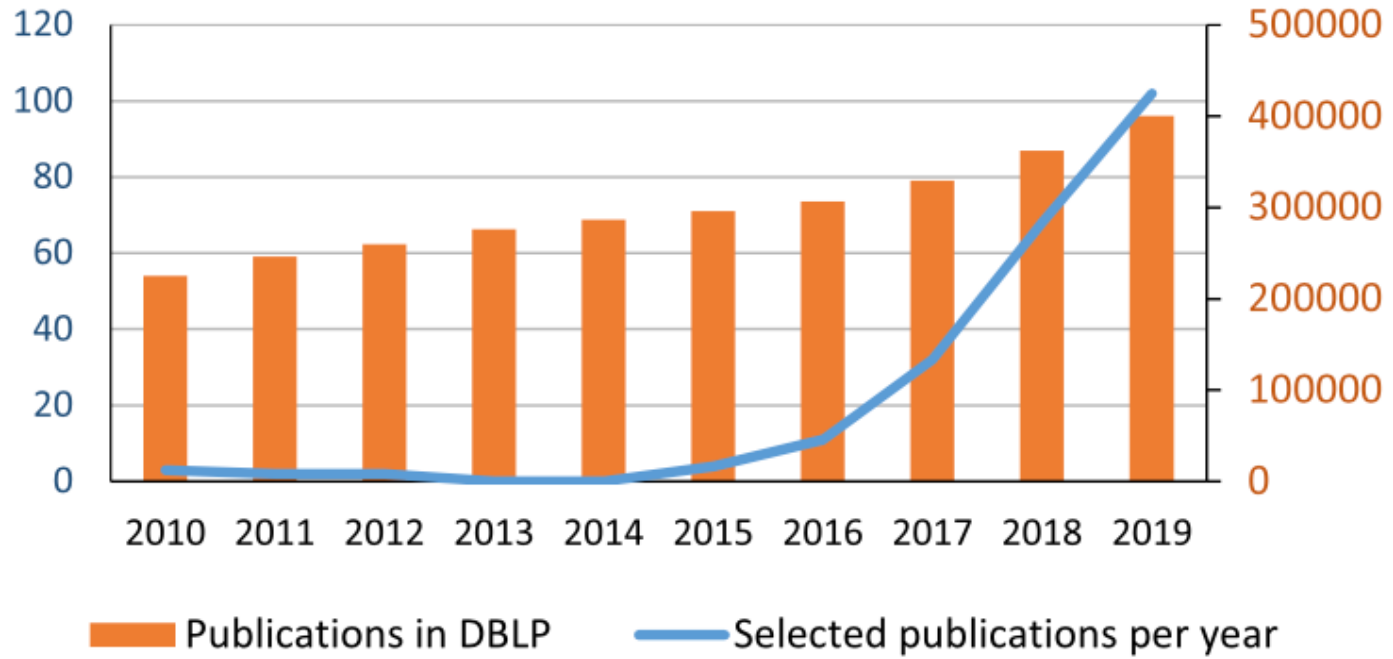
- Is good at modelling techniques and feature engineering
- Is mainly focused on accuracy
- Uses notebooks or similar for prototyping
- Is not interested in model size, updateability, implementation stability

Software engineer

- Builds products
- Concerned about cost, performance, stability, safety, security, and release time
- Must scale solution and handle large amounts of data
- Maintains, evolves, and extends the product over long periods

by Christian Kaestner

# NUMBER OF PUBLICATIONS

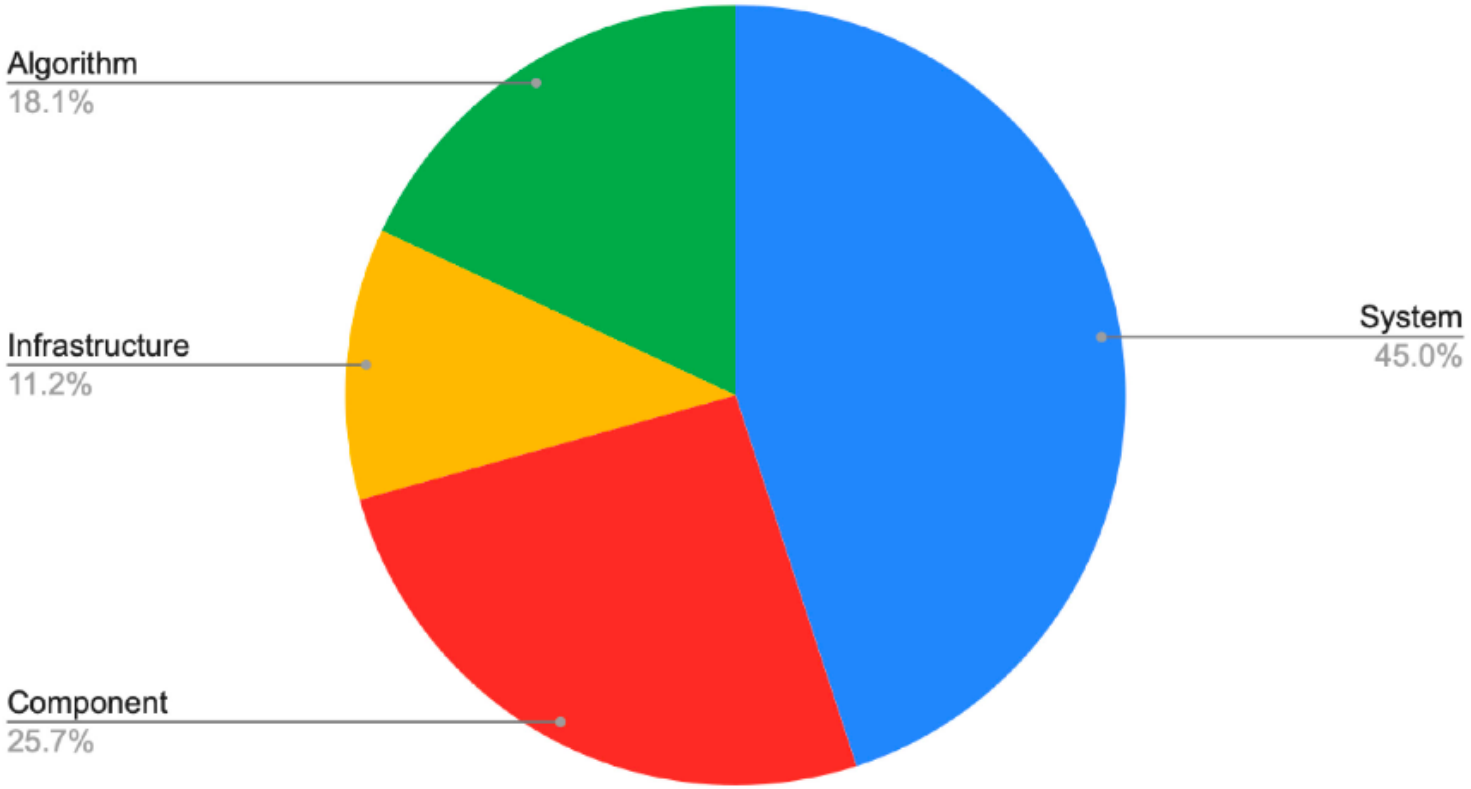



Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, Stefan Wagner: Software Engineering for AI-Based Systems: A Survey. ACM Trans. Softw. Eng. Methodol. 31(2): 37e:1-37e:59 (2022)

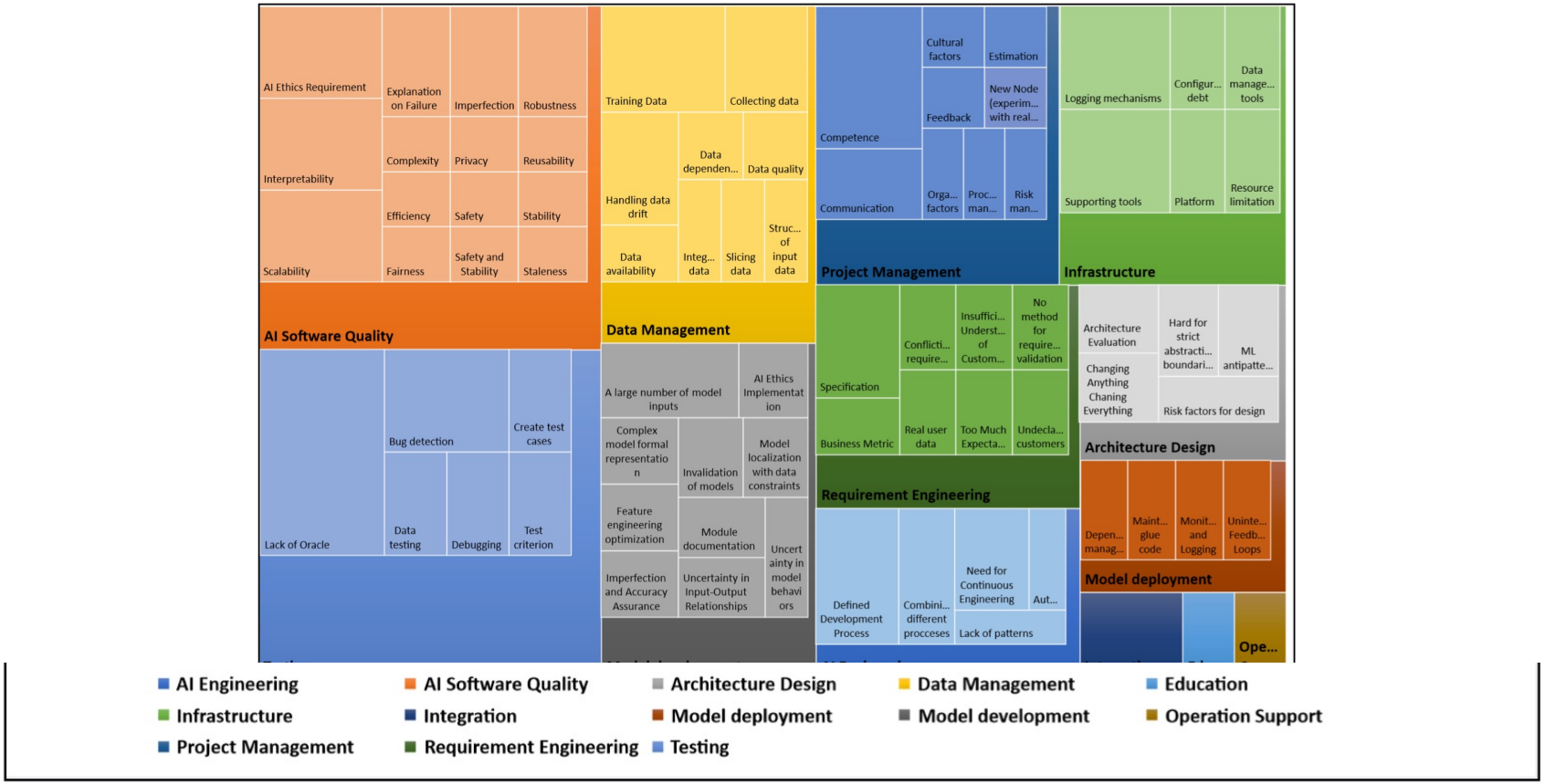
# AI-BASED SYSTEM PROPERTIES AND SE APPROACHES

| Study | Bibliometrics | AI-based systems properties | SE approaches | Challenges |
|---|---|---|---|---|
| [134] | 101 studies (2005–2018) | safety, correctness | practices for the evaluation and improvement of the software quality of ML applications | ML quality assurance |
| [211] | 38 studies (2008–2019) | | 33 unique architecture and design patterns for ML systems | |
| [23] | 64 studies (2002–2007, 2013–2016) | safety, robustness, reliability | verification and validation techniques for safety-critical automotive systems | verification and validation in DNNs |
| [27] | 37 studies (2012–2018) | correctness | testing practices | 18 challenges organized in 6 dimensions: implementation issues, data issues, model issues, written code issues, execution environment issues, mathematical design issues |
| [166] | 70 studies (2004-2019) | fairness, accuracy, safety, consistency | functional testing, test case generation and test oracle, integration testing, system testing | 11 challenges |
| [226] | 138 studies (2007-2019) | correctness, model relevance, robustness, security, data privacy, efficiency, fairness, interpretability | testing workflow, testing components, testing properties, application scenarios | 4 testing challenges categories: test input generation, test assessment criteria, oracle problem, testing cost reduction |
| [180] | 21 studies (2017–2019) | | 29 best practices for ML systems in six categories: data, training, coding, deployment, team, governance | |
| [208] | 15 studies (out of 906) (2009–2018) | | Model Evaluation, Deployment | |
| [115] | 115 studies (2003–2019) | safety, security, ethics and regulation, software structure, testability, maintainability, performance, risk and | all SWEBOK areas | SE challenges for ML applications |

# SCOPE OF RESEARCH



# MAIN CHALLENGES FACED IN THE DEVELOPMENT OF AI/ML SYSTEMS




Elizamary Nascimento, Anh Nguyen-Duc, Ingrid Sundbø, Tayana Conte: Software engineering for artificial intelligence and machine learning software: A systematic literature review. CoRR abs/2011.03751 (2020)

# SOME TAKEAWAYS

Empirical studies report AI software in various application domains, with the main focus being Automotive, Finance, and Healthcare

Testing space for AI software is much larger, more heterogeneous and, in many cases, it is difficult to formally define in comparison to traditional software testing

AI development processes need to integrate infrastructures, processes and tools for managing data as their integral parts

# ML

… is everywhere

The talk is not about new ML algorithms and solutions

It is about what we did and what we are doing

- It is also a way to think of our work in terms of ML

# ML SYSTEMS

They must offer predictions with given accuracy and precision

Usual requirements

- Response time is affected by resource allocation
- Computations can be executed on dedicated hardware (e.g., GPUs)
- Parallelization is not useful if data are not properly partitioned

New requirements

- Model quality also depends on the (hyper)-parameters of the learning algorithm
- Response time and quality are often correlated
  - Fewer learning iterations allow for faster results but produce less accurate models
- Results depend on used data

# TWO PHASES

## Training

- Long-lasting batch activity (from several minutes to days)
- Based on dedicated frameworks (e.g., Spark, TensorFlow, PyTorch)
- Maximum time to complete a training process

## Inference

- Interactive activity that exploits generated models
- Each computation lasts milliseconds or seconds
- Response time of multiple requests aggregated and constrained
- Quality of predictions cannot be easily computed

# KNOWN LIMITATIONS (TRAINING)

Available frameworks do not support quality and time constraints on carried out activities

Hyperparameters are defined at design time

- They must be carefully tuned to train the model efficiently and precisely

Data partitioning can accelerate the computation at the cost of more synchronization among executors

Resource allocation is key to fulfill time-based requirements

- Proactive vs reactive solutions
- Executors must be able to govern heterogeneous hardware

# KNOWN LIMITATIONS (INFERENCE)

ML frameworks lack support for the specification of quality and time requirements

Dynamic resource allocation is mandatory when the incoming workload fluctuates

- Requests must be scheduled to proper executors according to their hardware capabilities and the state of the system

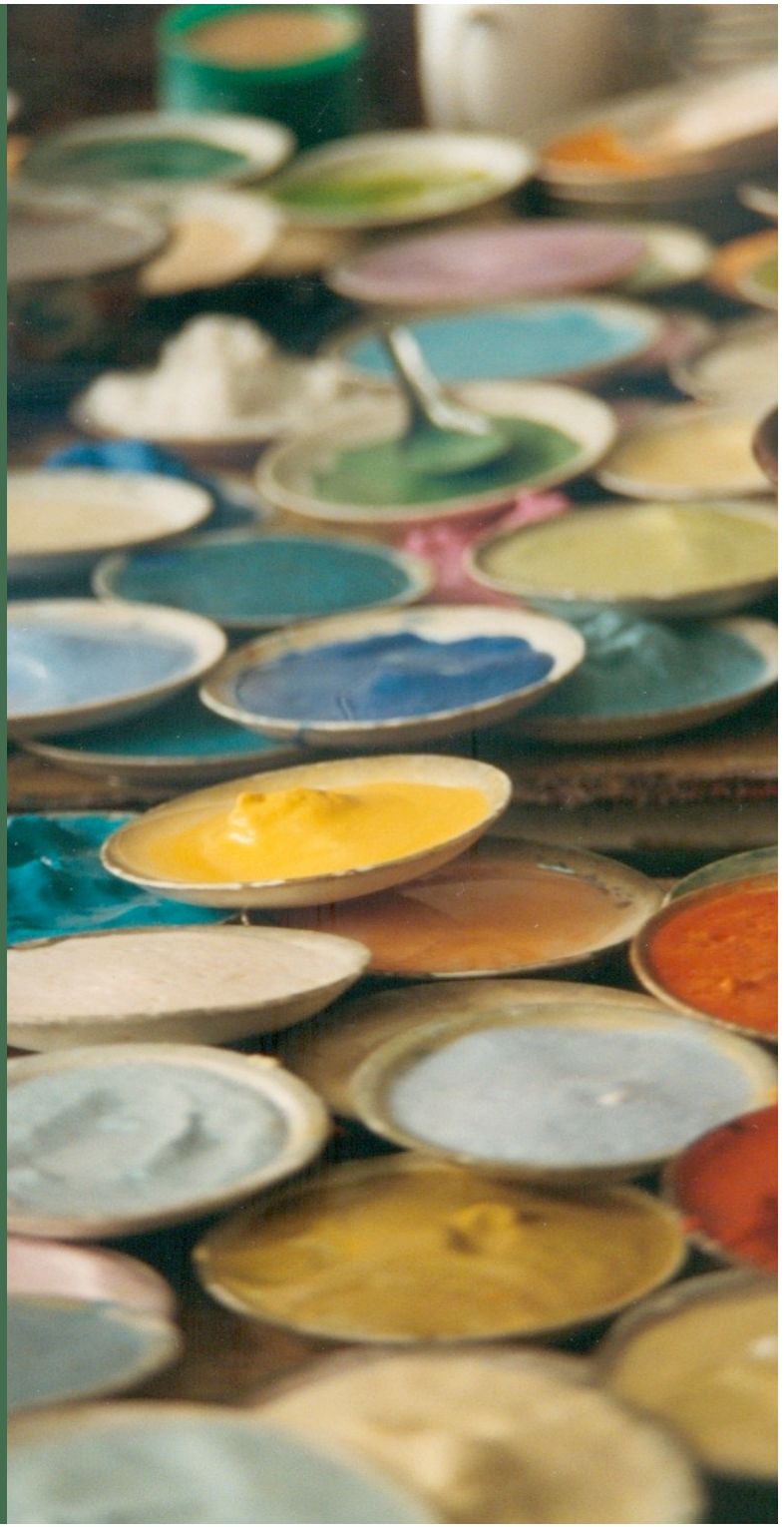
The quality of ML models at runtime cannot be directly monitored

- Uncertainty can help estimate it and understand when a model must be re-trained

# HOW HAVE WE TACKLED THESE ISSUES?

FSE 2016
TSE 2021
ICSOC 2021

From the original presentations by G. Quattrocchi

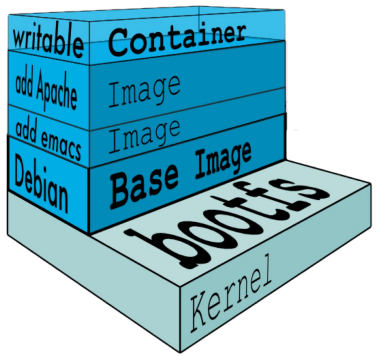


# TWO ENABLERS

*In collaboration with Prof. Alberto Leva

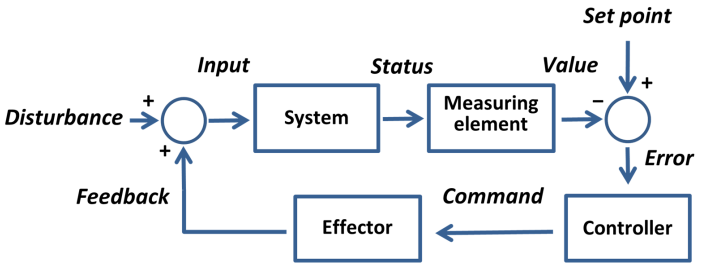
## Containers (Docker)

- Fast actuation
- Precise actuation
- Better resource utilization




## Control Theory*

- Fast decisions
- Precise decisions
- A priori guarantees



# DISTRIBUTED CONTROL OF CONTAINERS

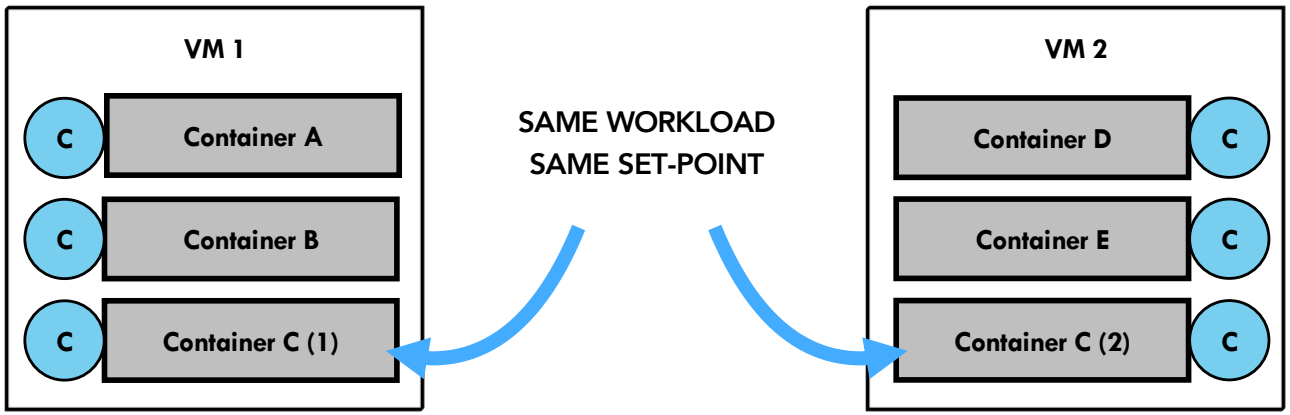
One controller per container

Proportional-Integral (PI) controllers

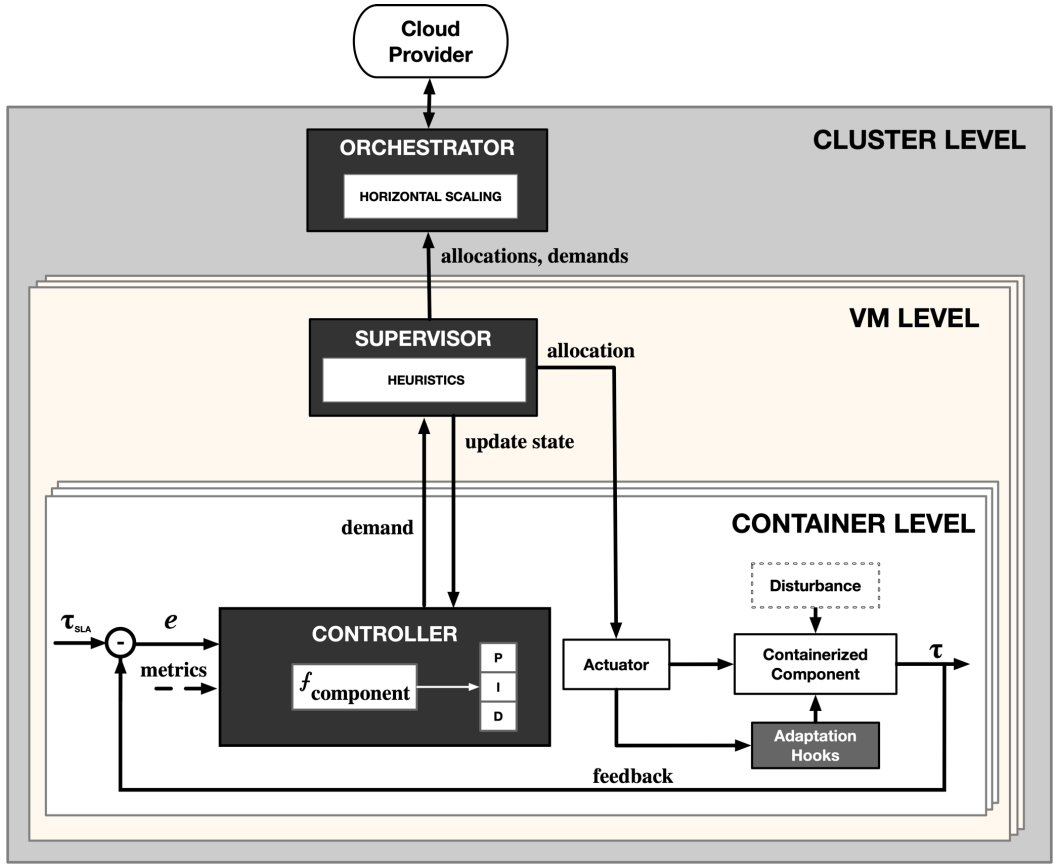
- Next state computed in constant time

Workload split equally to container replicas

- No synchronization is required



# ARCHITECTURE



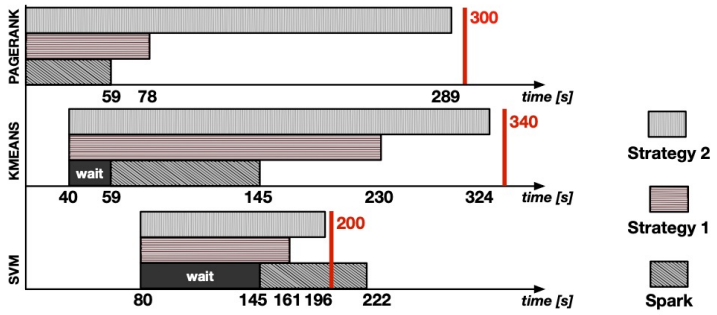
# DYNASPARK

DynaSpark extends Spark by introducing advanced and automated resource management

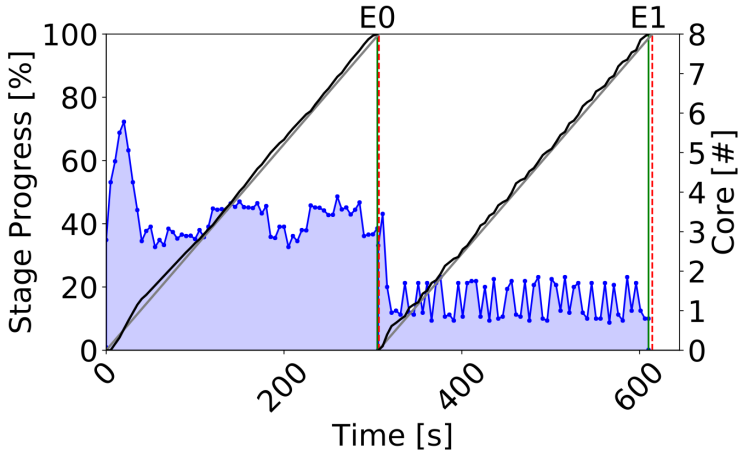
- Vertical CPU scaling (CPU quotas)
- Resource contention among running applications
- Different strategies (e.g., Earliest Deadline First) to prioritize applications

Prototype deployed on MS Azure

- Precision (how close we reach the deadline)
- The closer it gets to a deadline the fewer resources are used




Precision > 98%



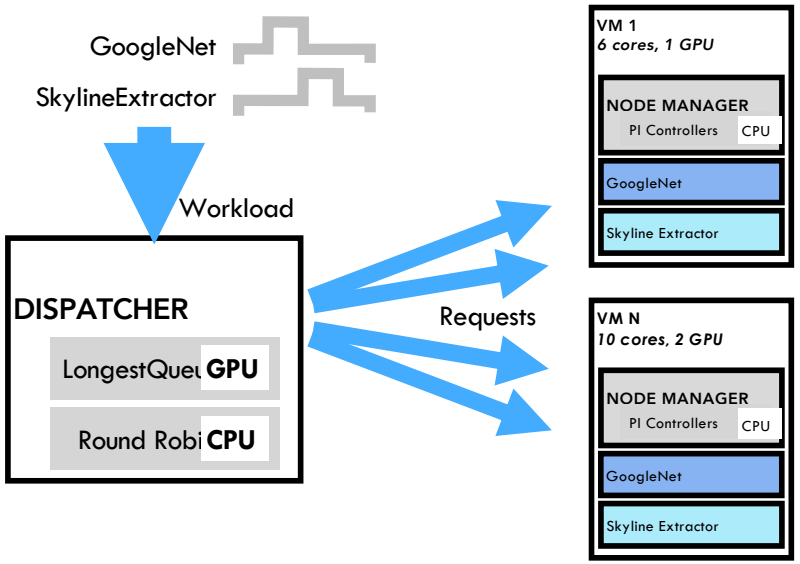
# ROMA

A solution to manage scientific GPU-empowered interactive applications

- Manages a cluster of heterogenous resources
- Targets interactive apps that can be accelerated with GPUs

GPUs are the preferred executors

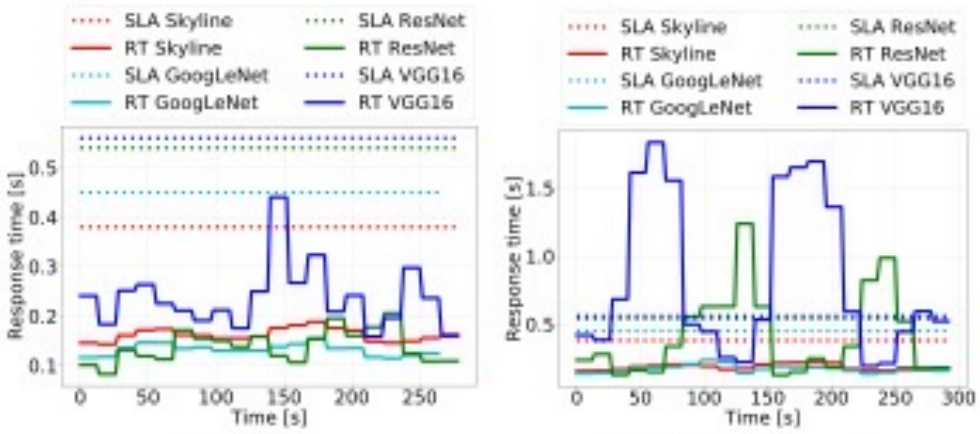
- CPUs are used only if GPUs are not enough
- Evaluated on TensorFlow



# EVALUATION

Prototype deployed on MS Azure

Metrics: number of violations and  amount of used resources




(a) *ROMA*

(b) Rule-based approach

**Violations**

20 times fewer
on average

**Resources**

32% fewer resources
on average

# FEDERATED MACHINE LEARNING AS A SELF-ADAPTIVE PROBLEM

SEAMS 2021

From the original presentation by G. Quattrocchi

# HYPERFL

Federated (Machine) learning as a self-adaptive system

An extension to TensorFlow to allow a set of federated nodes to cooperatively train an ML model under quality constraints

- Minimise Training Time
- Minimise Resource Consumption
- Maximise Model Quality (e.g., Accuracy)
- Minimize Used Bandwidth

# MACHINE LEARNING

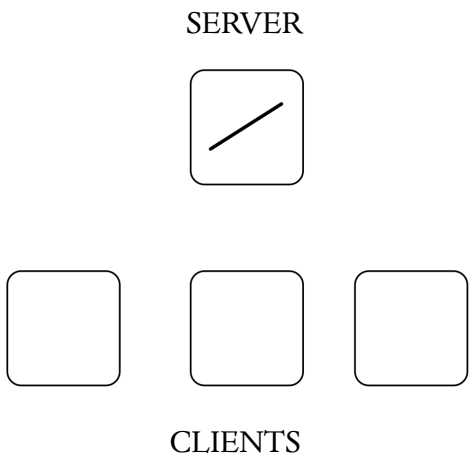
Clients send data to a centralized server (or a cluster of servers)

- A model is trained on the server
- Clients use the model to carry out inference

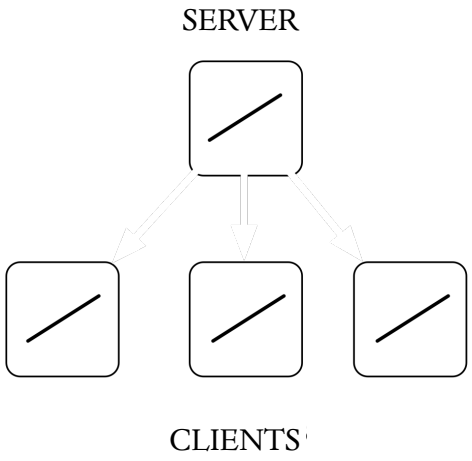
Problems

- Privacy: clients have to share private data
- Network overhead: all raw data must be sent to the server

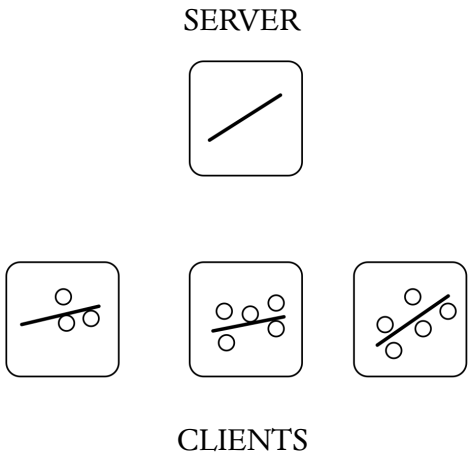
# FEDERATED MACHINE LEARNING (FEDML)




The centralised server initialises a model with some weights

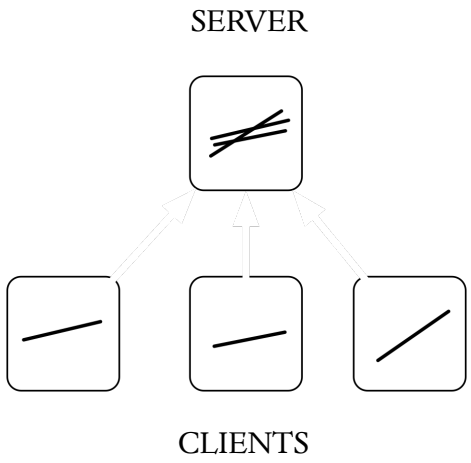



The model is sent to the clients

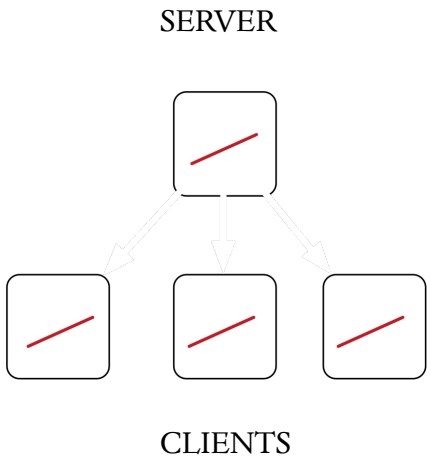



Each client performs an independent iterative training using only local data

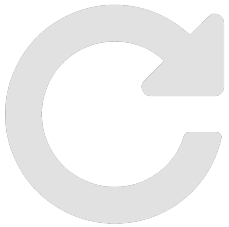
# FEDERATED MACHINE LEARNING (FEDML)




When clients finish the local training they send the updated weights (and no data) to the server




The server aggregates the results and computes a new model that is sent to all clients



This process is repeated multiple times (rounds)

# MONITORING AND ANALYSIS

CPU/Memory consumption

Battery Level

Accuracy, Loss (calculated in a federated fashion)

Client training time

Total training time

Network overhead

# PLANNING AND EXECUTION

## Client Selection (server-side only)

- Decides which clients to involve in each round
- E.g., depending on availability, client resources

## Data Selection (server/client side)

- Decides the size of the training dataset of each client at each round
- Decides the model weights that each client must send to server at each round

## Workload and Resource Allocation (server/client side)

- Number of iterations (e.g., epochs) to be performed by each client at each round
- Resources to be used by each client at each round

# FORMULATION

TABLE I: Formulation of the generalized FedML problem.

**Constants**

| | |
|---|---|
| $C$ | Set of all clients |
| $W$ | Model weights to be trained |
| $A_c^r$, | User data available on each client $c$ at each round $r$ |
| $X_c$, | Resources of each client $c$ |
| $TT_{SLA}$ | Training time threshold |
| $AC_{SLA}$ | Accuracy threshold |
| $UB_{SLA}$ | Used bandwidth threshold |
| $RC_{SLA}$ | Resource consumption threshold |

**Parameters**

| | |
|---|---|
| $R$ | Total number of rounds |
| $K^r$ | Fraction of clients involved in training at each round $r$ |
| $E_c^r$ | Number of epochs for each round $r$ for each client $c$ |
| $B_c^r$ | Batch size for each round $r$ and client $c$ |
| $N_c^r$, | Data used for each training round $r$ and client $c$ |
| $S_c^r$ | Selected clients at each round $r$ |
| $V_c^r$ | Fraction of model weights sent by each client $c$ at round $r$ |
| $U_{c,w}^r$ | Selected weights $w$ at each round $r$ and by client $c$ |

**Metrics**

$TT(R, S, X, \frac{E \cdot N}{B}, C \cdot K, W \cdot V)$
$RC(R, S, X, \frac{E \cdot N}{B}, C \cdot K)$
$AC(R, K, S, \frac{E \cdot N}{B}, V, U, \eta)$
$UB(R, C \cdot K, W \cdot V)$

**Fixed Constraints**

**C1.** $0 < K^r \leq 1, K^r \in \mathbb{R}, \forall r \in R$
**C2.** $0 \leq V_c^r \leq 1, V_c^r \in \mathbb{R}, \forall r \in R, \forall c \in C$
**C3.** $0 < B_c^r \leq N_c^r, B_c^r \in \mathbb{R}, \forall r \in R, \forall c \in C$
**C4.** $E_c^r \geq 0, E_c^r \in \mathbb{N}, \forall r \in R, \forall c \in C$
**C5.** $\sum_{c=0}^{C} S_c^r = K^r \cdot C, \forall r \in R$
**C6.** $N_c^r < A_c^r, \forall r \in R, \forall c \in C$
**C7.** $\sum_{w=0}^{W} U_{c,w}^r = V_c^r \cdot W, \forall r \in R, \forall c \in C$

**Metric-related Constraints**

**C8.** $RC(R, S, X, \frac{E \cdot N}{B}, C \cdot K) \leq RC_{SLA}$
**C9.** $UB(R, C \cdot K, W \cdot V) \leq UB_{SLA}$
**C10.** $TT(R, S, X, \frac{E \cdot N}{B}, C \cdot K, W \cdot V) \leq TT_{SLA}$
**C11.** $AC(R, K, S, \frac{E \cdot N}{B}, V, U, \eta) \geq AC_{SLA}$

**Problems**

**P1.** $minimize\ RC(R, S, X, \frac{E \cdot N}{B}, C \cdot K)$ *subject to* $C11$
**P2.** $minimize\ RC(R, S, X, \frac{E \cdot N}{B}, C \cdot K)$ *subject to* $C9, C11$
**P3.** $minimize\ RC(R, S, X, \frac{E \cdot N}{B}, C \cdot K) \wedge$
$minimize\ UB(R, C \cdot K, W \cdot V)$ *subject to* $C10$
**P4.** $minimize\ TT(R, S, X, \frac{E \cdot N}{B}, C \cdot K, W \cdot V) \wedge$
$minimize\ UB(R, C \cdot K, W \cdot V)$ *subject to* $C11$
**P5.** $maximize\ AC(R, K, S, \frac{E \cdot N}{B}, V, U, \eta)$ *subject to* $C8$
**P6.** ...

Table I summarises our formulation[2]. Let us start from minimum or maximum thresholds for our metrics: $TT_{SLA}$ for total training time, $AC_{SLA}$ for accuracy, $UB_{SLA}$ for used bandwidth, and $RC_{SLA}$ for resource consumption.

In addition to constants $C$ and $W$, we define $A_c^r$, an $R \times C$ matrix that contains the amount of examples (e.g., images) on each client at each round, and $X_c$ as the $C$-sized vector that contains the amount of resources available on each device. Instead of always considering the same amount of clients $K$, we define an $R$-sized vector $K^r$ to consider a different number of devices at each round. Similarly, $E_c^r$ and $B_c^r$ ($R \times C$ matrices) define the amount of epochs and size of local batches to allow one to tweak these values at each round and on each client. $N_c^r$ is the amount of data used for training at each round and for each client ($R \times C$ matrix). Note that the original formulation used all available data on a client for training ($N_c = A_c$) and $N_c$ was constant for the whole training. The $R \times C$ matrix $S_c^r$ defines the $K$ clients employed at each training round (the matrix contains 1 if client $c$ is active at round $r$; 0 otherwise).

While the original formulations always send all locally-computed weights to the server at each round, the transmission of a significant subset could be more efficient [10]. To this end, $V_c^r$, an $R \times C$ matrix, defines the fraction of $W$ parameters sent at each round by each client. $U_{c,w}^r$, an $R \times C \times W$ matrix, defines $(0/1)$ the model parameters sent by each client at each round.

The four performance indicators (metrics) are formally defined as (unknown) functions that depend on the aforementioned constants and parameters. All the metrics depend on the number of rounds $R$ and amount of selected clients $K \cdot C$, device selection $S$ affects resource consumption, training time and accuracy, available resources $X$ impact resource consumption and training time, the amount of local updates $O$ impacts all the metrics except used bandwidth, while the selection of weights ($V$ and/or $U$) impacts training time, accuracy, and used bandwidth.

The minimization/maximization of these functions must also satisfy a set of constraints. The constraints are grouped in two sets: the first one ($C1$-$C7$) defines parameters' domains and their relationships with one another. Constraints $C1$-$C4$

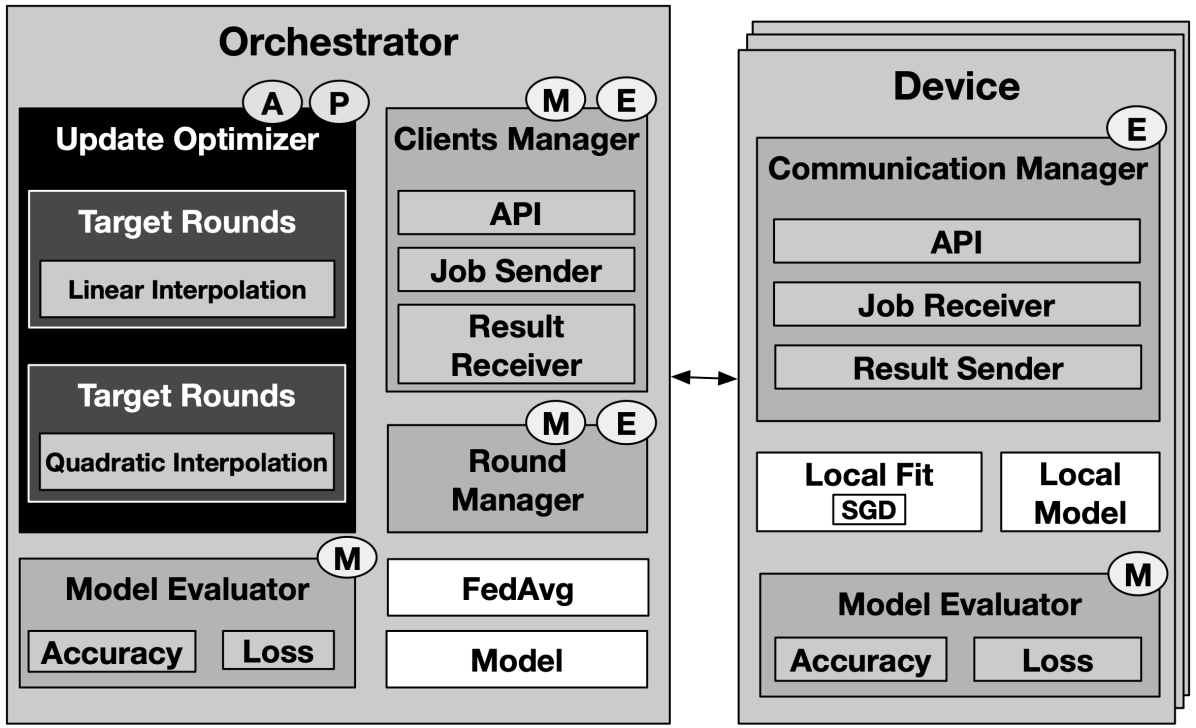
# PROTOTYPE

GOAL: minimize resource consumption given a constraint on the accuracy in a given number of rounds

Workload selection (number of iterations or epochs)

- Server-side planning
- Heuristic: linear and quadratic interpolation

# PROTOTYPE


Orchestrator
A
P
Update Optimizer
Target Rounds
Linear Interpolation
Target Rounds
Quadratic Interpolation
M
E
Clients Manager
API
Job Sender
Result Receiver
M
E
Round Manager
M
Model Evaluator
Accuracy
Loss
FedAvg
Model
Device
E
Communication Manager
API
Job Receiver
Result Sender
Local Fit
SGD
Local Model
M
Model Evaluator
Accuracy
Loss

# SETUP

We implemented the prototype in a custom simulation environment

- TensorFlow
- 50 clients

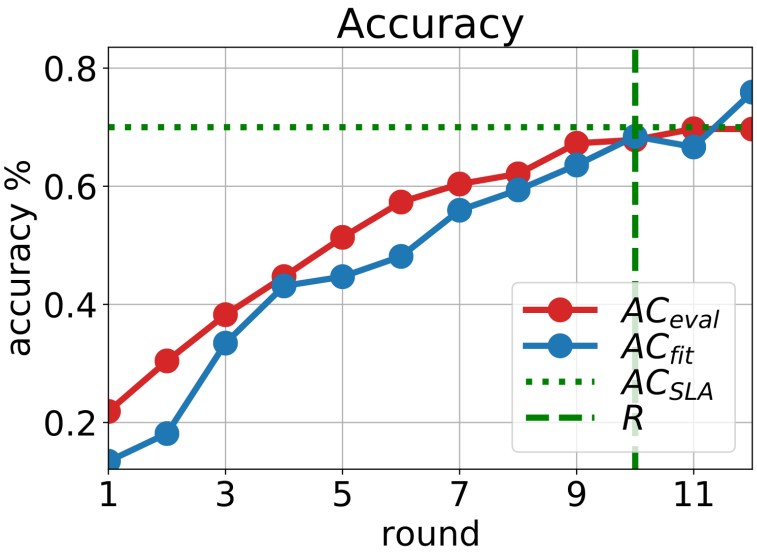
Two real-world applications: MNIST and Fashion-MNIST

- 115 experiments

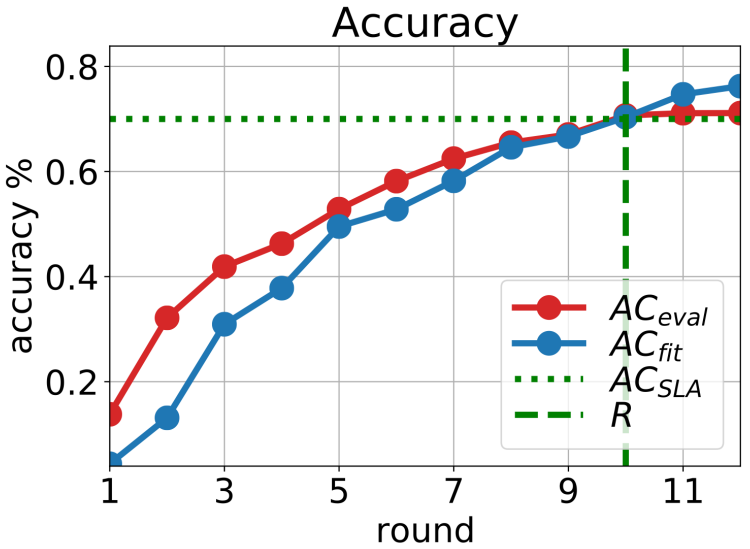
# RESULTS

APPLICATION: Fashion-MNIST

SETPOINT: Accuracy = 70% in 10 rounds




Linear Interpolation




Quadratic Interpolation

# CONCLUSIONS

FEDML systems call for self-adaptation

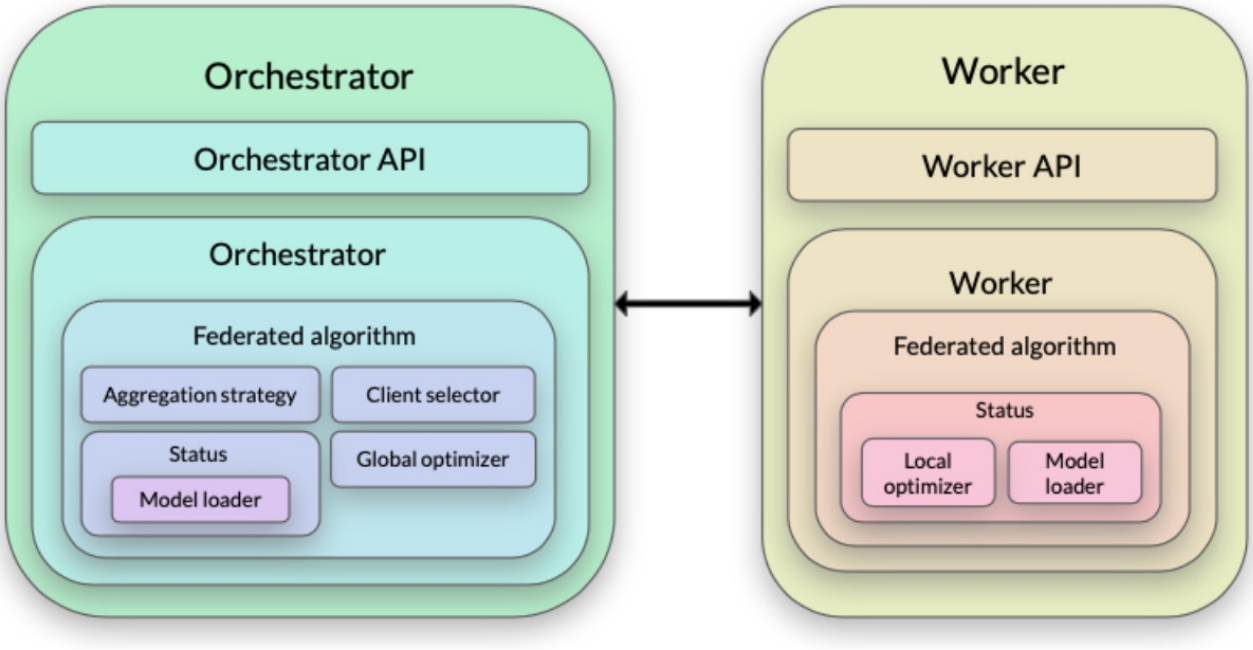
Control loops could be deployed in the centralized server and in the federated clients

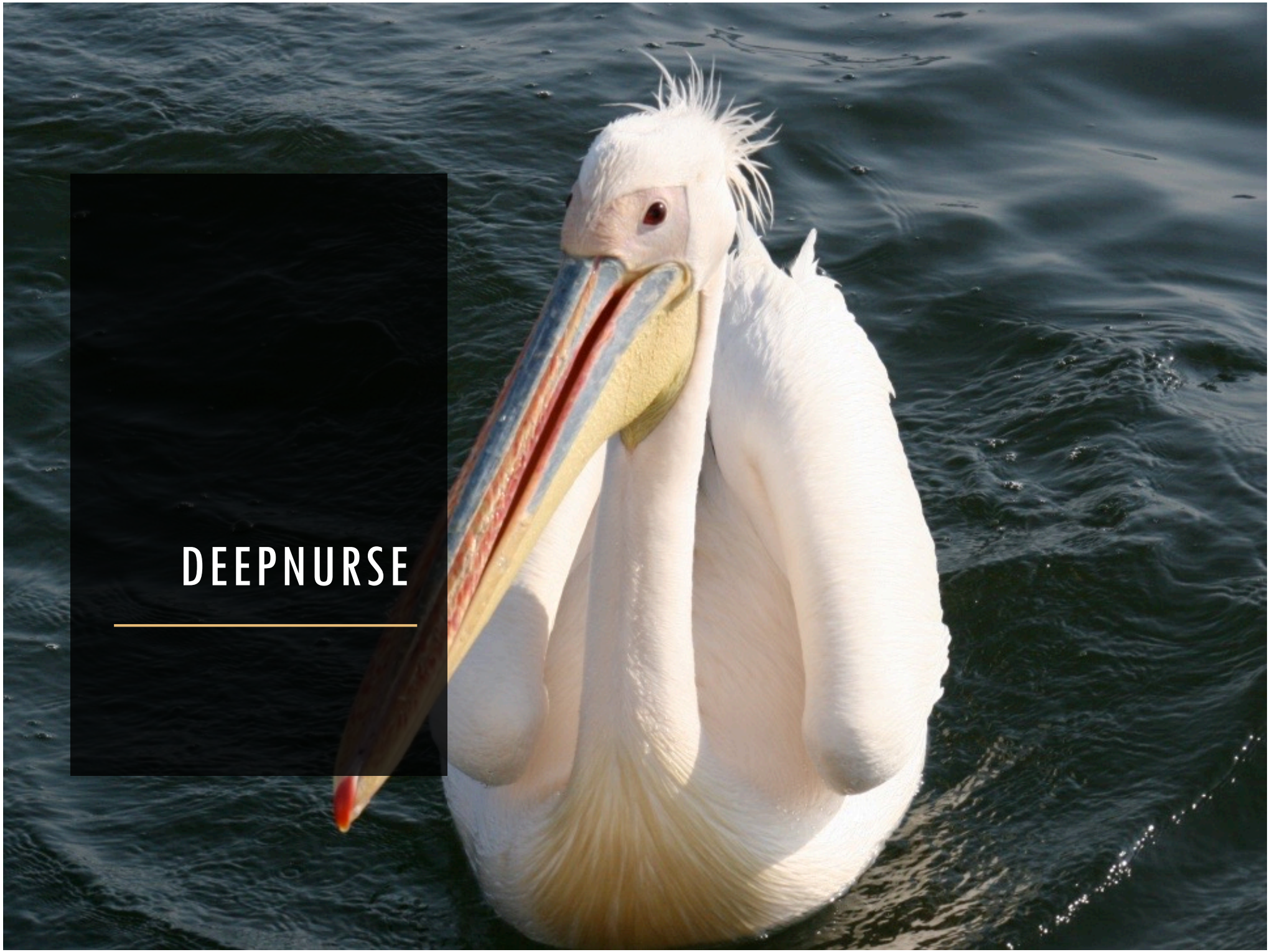
Initial results are promising

Future work

- Client selection
- Fine-grained client-side control
- Creation of a real FEDML framework (in progress)
- P2P FEDML (just initiated)

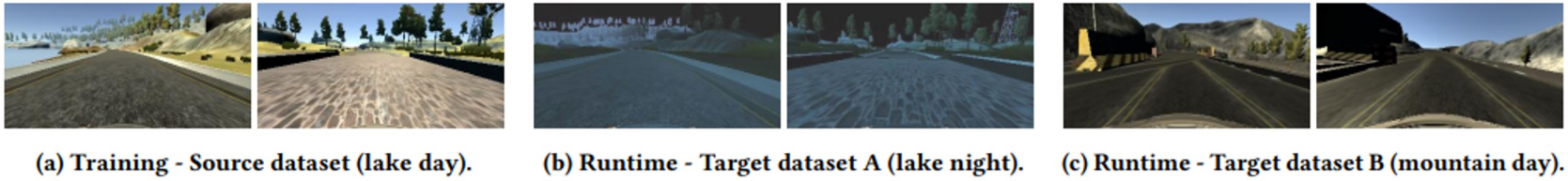
# FELES: A FEDML SIMULATOR




DEEPNURSE

# PROBLEM

Neural network accuracy decreases when dealing with data from unknown domains (Out-Of-Distributions)



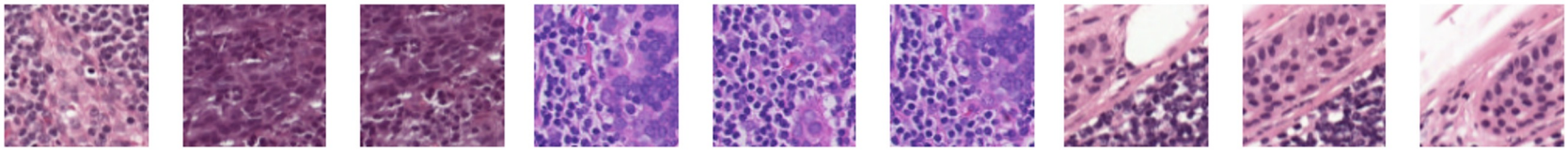
(a) Training - Source dataset (lake day).



(b) Runtime - Target dataset A (lake night).



(c) Runtime - Target dataset B (mountain day).



(d) Training - Source dataset (hospital A).



(e) Runtime - Target dataset A (hospital B).



(f) Runtime - Target dataset B (hospital C).

# DEEPNURSE

Four-stage loop that automatically detects unknown domains and adapts neural networks to them

- Uncertainty estimation
- Changepoint detector
- Style-guided data generation
- Weights adaptation

# UNKNOWN DOMAIN DETECTION

Uncertainty estimation with Bayesian neural networks

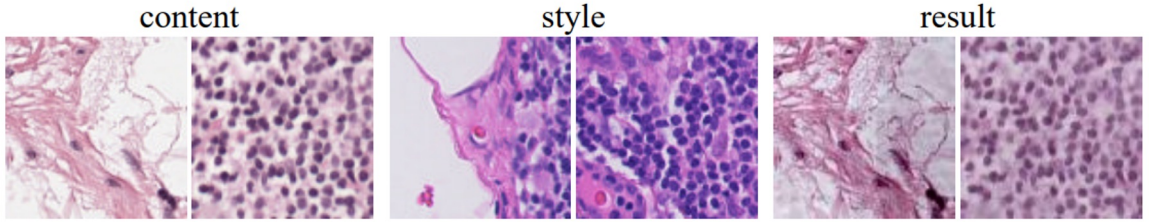
- Deep Ensemble
- MonteCarlo dropout

Changepoint Detector

- Sliding window
- Kolmogorov-Smirnov two sample test

# STYLE-GUIDED DATA GENERATION


**(a) Landscape and daytime style transfer (lake day→mountain night).**




**(b) Hospital style transfer (hospital A→hospital B).**

# ADAPTATION TO UNKNOWN DOMAINS

## Weights adaptation

- Reset (Retraining from scratch)
- Partial (Retraining from trained model)
- Fine-tuning (Retraining only last layers)

## Incremental learning

- Replay memory to avoid catastrophic forgetting

# EXPERIMENTS

Three datasets

Classification

- Camelyon17 (Tumor Detection)
- Waterbirds (Bird Classification)

Regression

- Udacity* (Autonomous  Driving)

*Custom collected dataset with 16 different driving scenarios

# RESULTS

Eight different neural network architectures

- DenseNet, Resnet (classification)
- Dave-2, Epoch, Chauffeur (regression)

Results:

- Detected on average 81.0% of Unknown Domains
- Recovered 24.1% of the performance after adaptation


LET US FLIP THE PROBLEM

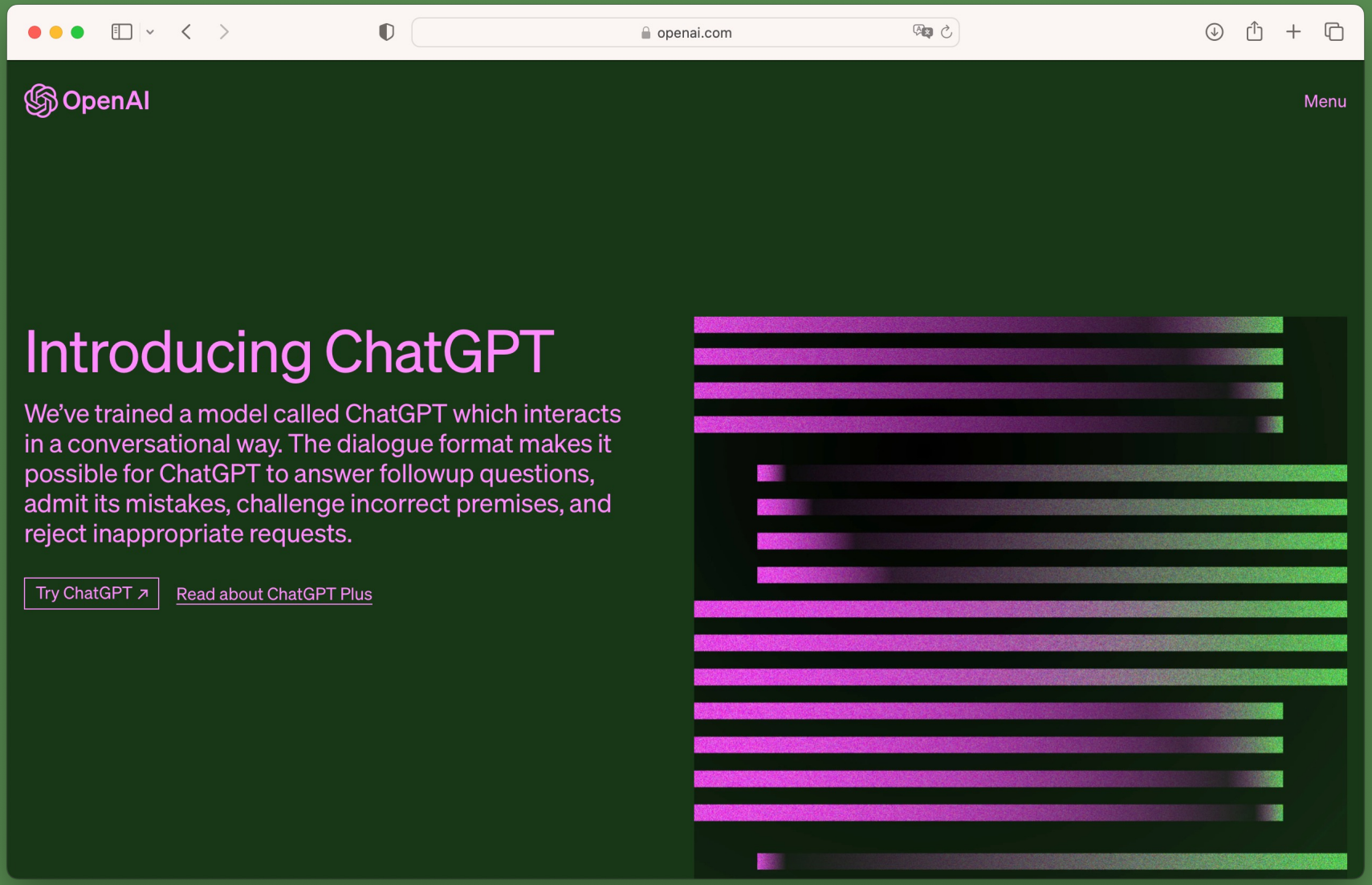
# AI FOR SOFTWARE ENGINEERING

Everywhere

Think of a software engineering problem and you can conceive an AI-based solution

Some examples

- Requirements management/elicitation
- Software design
- Test and analysis
- Software architecture
- Microservice/service/component composition
- Code repearing
- Code completion
- ….


openai.com
OpenAI
Menu
Introducing ChatGPT
We've trained a model called ChatGPT which interacts in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests.
Try ChatGPT ↗
Read about ChatGPT Plus

# FEATURE MODEL-GUIDED ONLINE REINFORCEMENT LEARNING FOR SELF-ADAPTIVE SERVICES

ICSOC 2020 (Best paper award)

From the original presentation by A. Metzger

# SELF-ADAPTIVE SERVICE

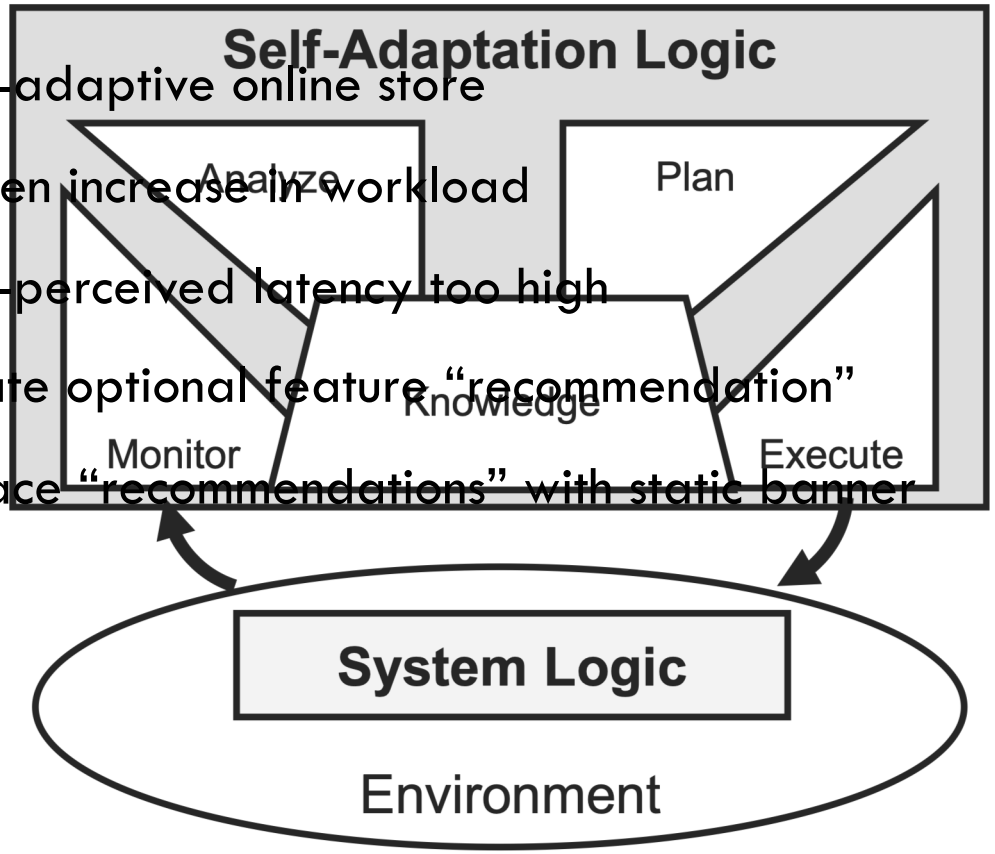
Example: Self-adaptive online store

Monitor: Sudden increase in workload

Analyze: User-perceived latency too high

Plan: Deactivate optional feature “recommendation”

Execute: Replace “recommendations” with static banner

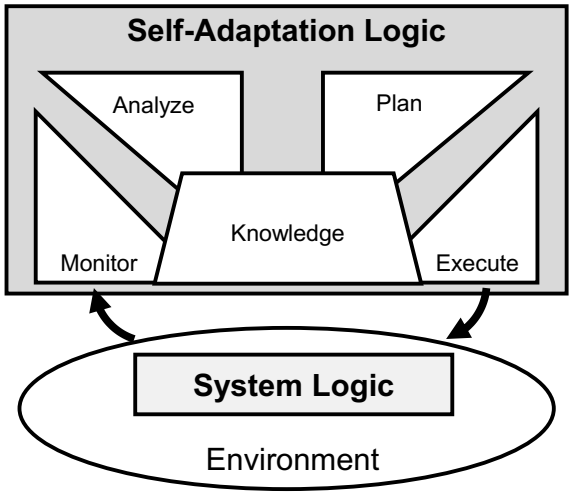


# "DESIGN TIME" UNCERTAINTY

Infeasible to anticipate all future environment situations (e.g., QoS of dynamically bound services)

Difficult to precisely determine the impact of adaptation actions on QoS (e.g., exact QoS impact when adding a VM)

Simplifying assumptions (e.g., too much effort to explicitly codify all details as knowledge)



# ONLINE REINFORCEMENT LEARNING (RL)

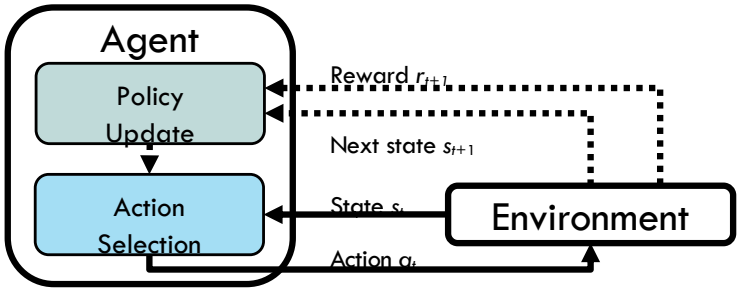
Learn suitable action selection policy via agent's interactions with environment

Agent receives reward for executing an action (here: adaptation action)

Reward expresses how suitable action was  (here: QoS satisfaction)

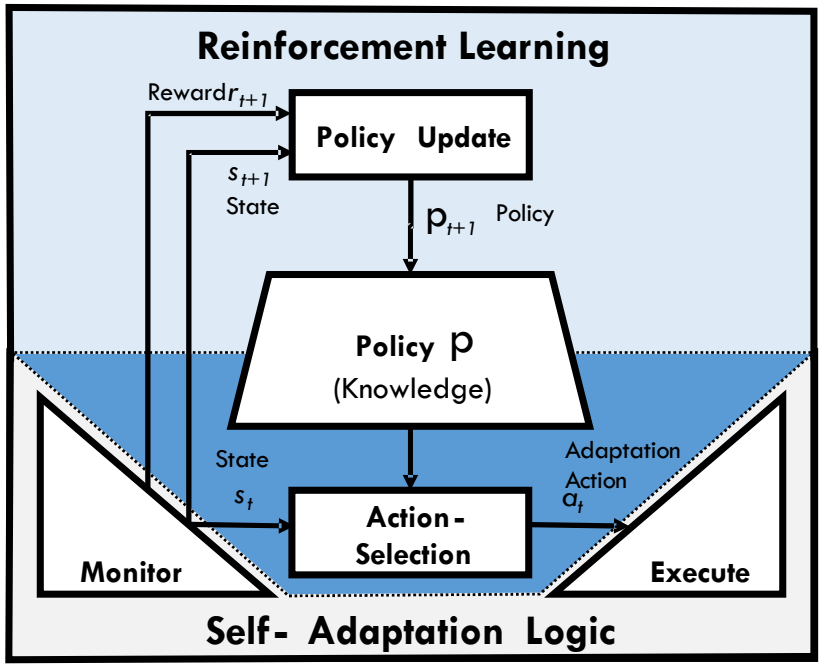
Update policy from reward signal = learn

Goal of RL: optimize cumulative rewards



# ONLINE RL FOR SELF-ADAPTIVE SERVICES

Combining MAPE-K and RL [Palm et al., 2020]



# PROBLEM STATEMENT

Exploitation-exploration dilemma of RL [Sutton & Barto, 2018]

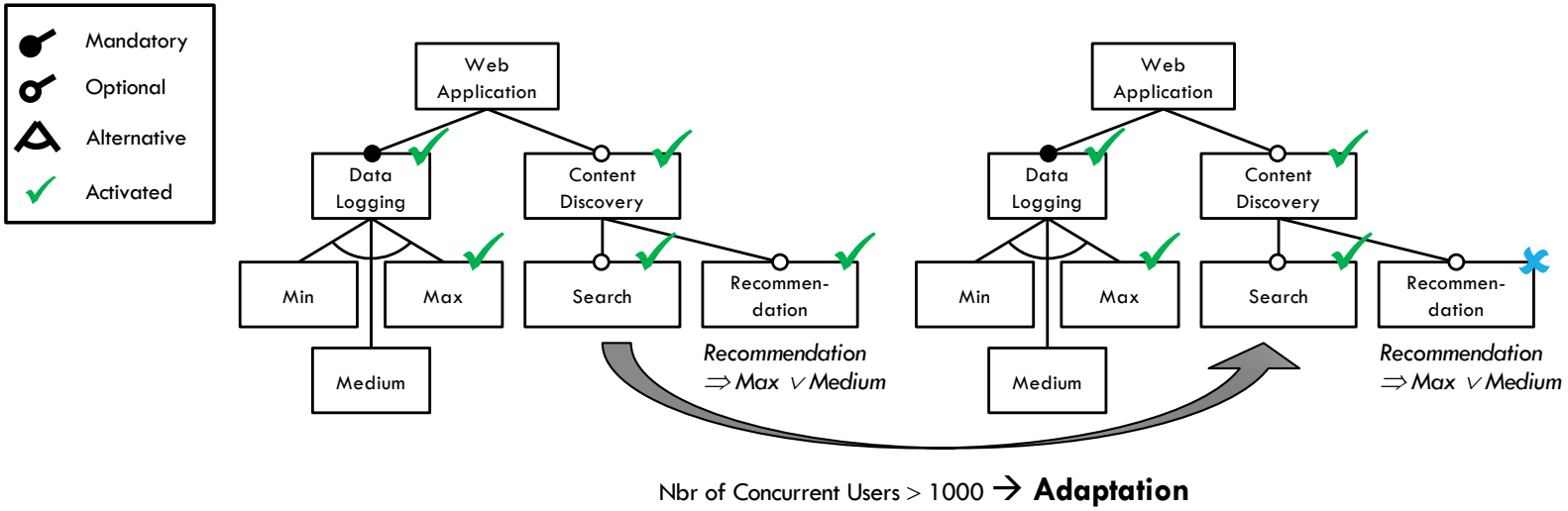
- Exploit existing knowledge vs explore new knowledge

How adaptation actions are explored impacts on learning performance

Limitations of State of the Art in RL for self-adaptive services

- (1) Random exploration (ε-greedy)
  - Slow learning if large set of adaptation actions
- (2) Evolution-unaware exploration
  - New adaptations explored with low probability and thus late

# FEATURE MODELS




Nbr of Concurrent Users > 1000 → **Adaptation**

Feature model expresses system configurations in compact form

Concrete system configuration expressed as feature combination

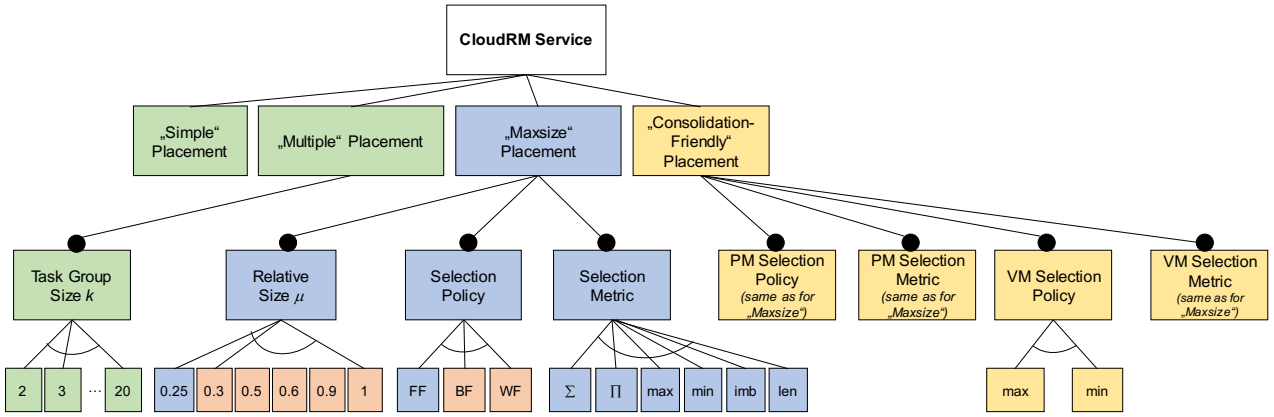
Adaptation expressed as runtime reconfiguration

# EXPERIMENT SETUP

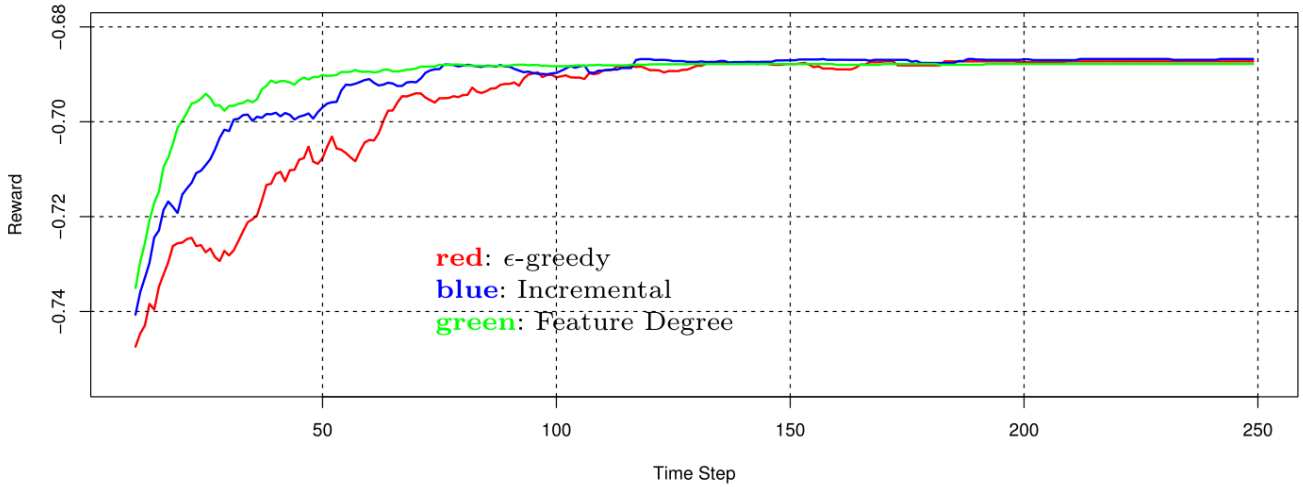
CloudRM – Self-adaptive Cloud Resource Management Service

- Feature Model (Defines 344 configurations = adaptation actions)
- Real-world workload trace
  - 10,000 tasks, 29 days
- Simulated Evolution of Adaptation Space

Initial
Evolution step #1
Evolution step #2
Evolution step #3



# (1) LARGE ADAPTATION SPACE




| | | | |
|---|---|---|---|
| Asymptotic performance | 0% | Energy savings | 0.1% |
| Time to threshold | 48.6% | Reduced VM migrations | 7.8% |
| Jumpstart | 1.3% | | |
| Total reward | 58.8% | | |

# MAIN RESULTS

Exploiting structural knowledge from design time (feature models) to guide online learning for self-adaptive services

Future enhancements

- Experiments with additional systems
- Comparison of other exploration strategies and RL algorithms
- Considering changes of existing features (on top of additions and removals)
- Methodology for defining suitable feature models during design time

# CONCLUSIONS

## Many different possible combinations

- Software engineering vs. Artificial Intelligent

## More to come

- Almost any paper submitted to SE conferences embeds AI or ML
- It would be nice to know what the others think of it

## Some ideas for the future

- Are we sure it is always the right way?
- Performance, quality, precision
- Ethical issues


THANK YOU !!!
(谢谢你)
luciano.baresi@polimi.it

# WE ARE HIRING !!!