Advanced Networked Systems SS24

Machine Learning for Networking

Prof. Lin Wang, Ph.D.
Computer Networks Group
Paderborn University
https://cs.uni-paderborn.de/cn

Learning objectives

How to leverage machine learning for **video streaming**?

How to leverage machine learning for **network packet classification**?

Machine learning paradigms

Supervised learning (regression, classification)	**Unsupervised learning** (clustering)	**Reinforcement learning** (decision making)

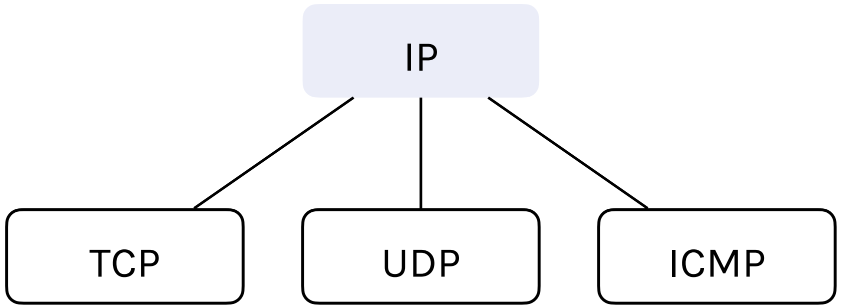
Where does **deep learning** sit? Well, deep learning is part of a broader family of machine learning methods that are based on artificial neural networks and can fit in any of the above categories.

Deep reinforcement learning in the spotlight

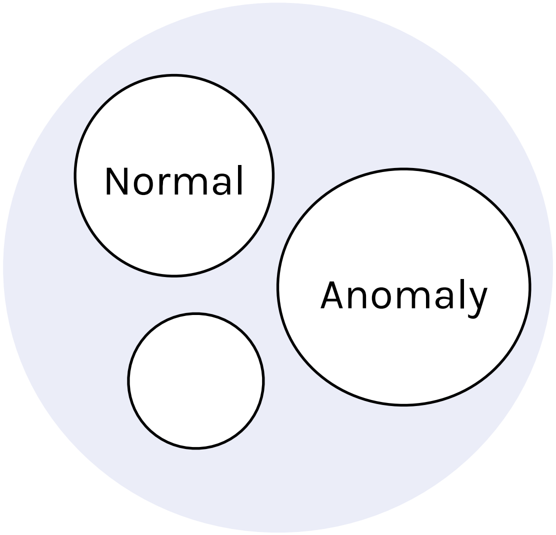
RL-based agent beat human experts on Go and more

https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go

ML in networking: examples



Decision tree for packet classification

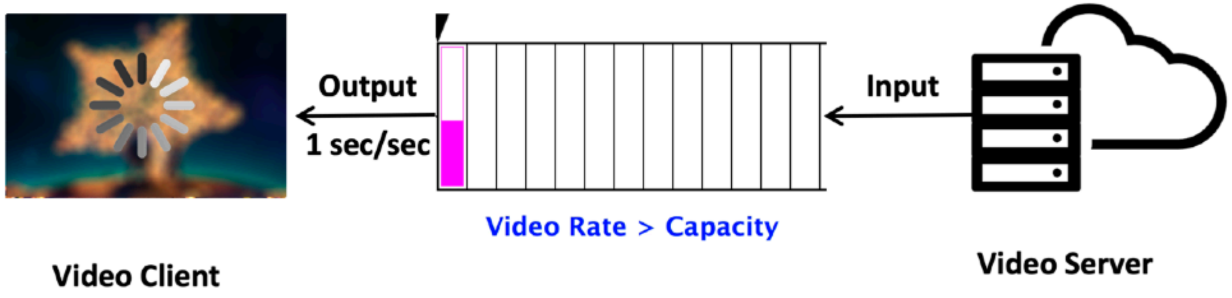


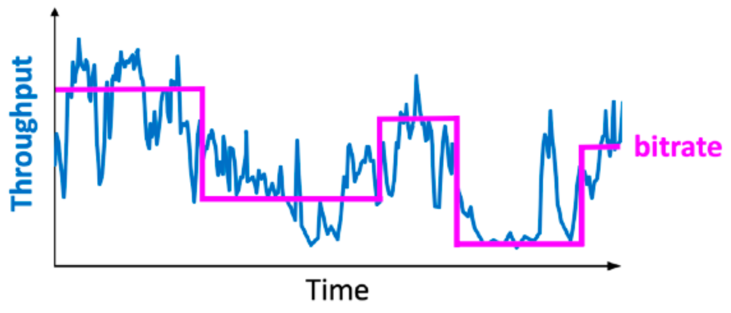
Clustering for anomaly detection

Machine Learning for Adaptive Video Streaming

Modern video streaming

Dynamic Streaming over HTTP (DASH)

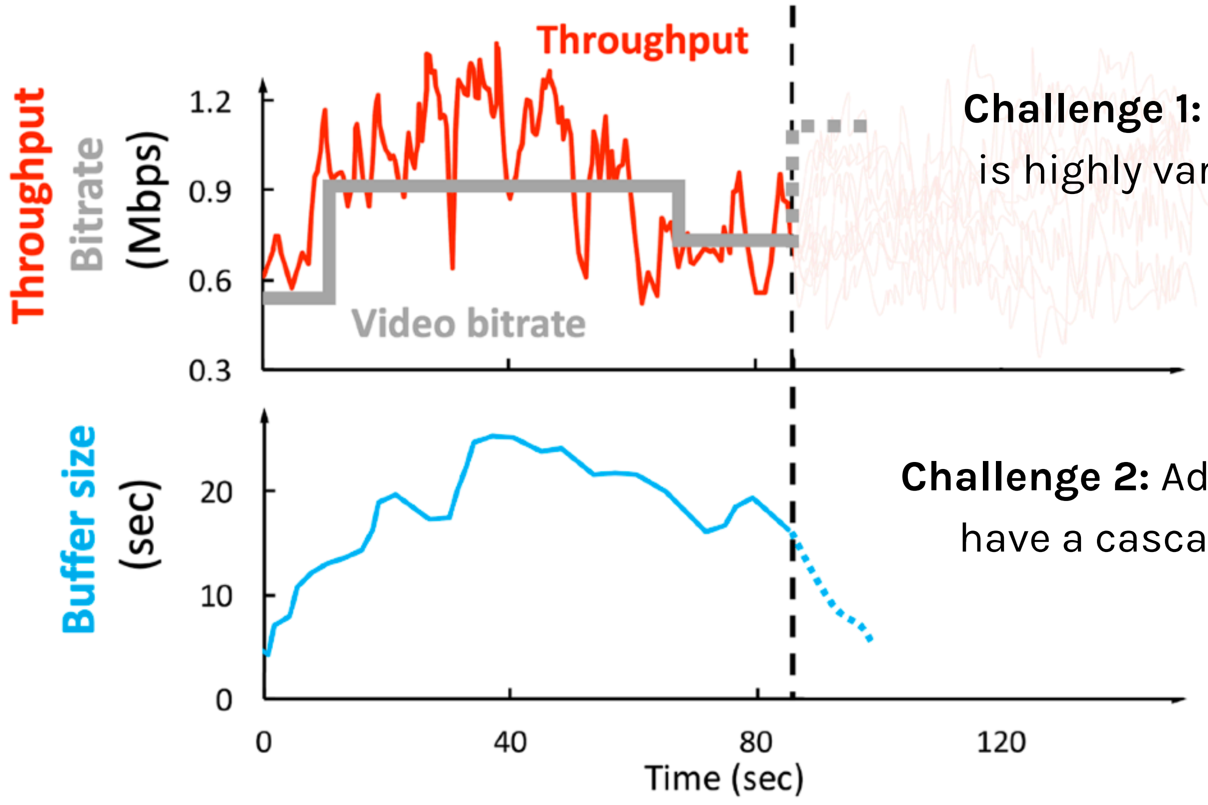




Adaptive bitrate (ABR) algorithms:

- Rate-based
- Buffer-based

Why is video streaming a challenging problem?



Challenge 1: Network throughput is highly variable and uncertain

Challenge 2: Adaptation decisions may have a cascading effect over time

Existing solutions

Rate-based: pick bitrate based on predicted throughput

- FESTIVE (CoNEXT'12), PANDA (JSAC'14), CS2P (SIGCOMM'16)

Buffer-based: pick bitrate based on buffer occupancy

- BBA (SIGCOMM'14), BOLA (INFOCOM'16)

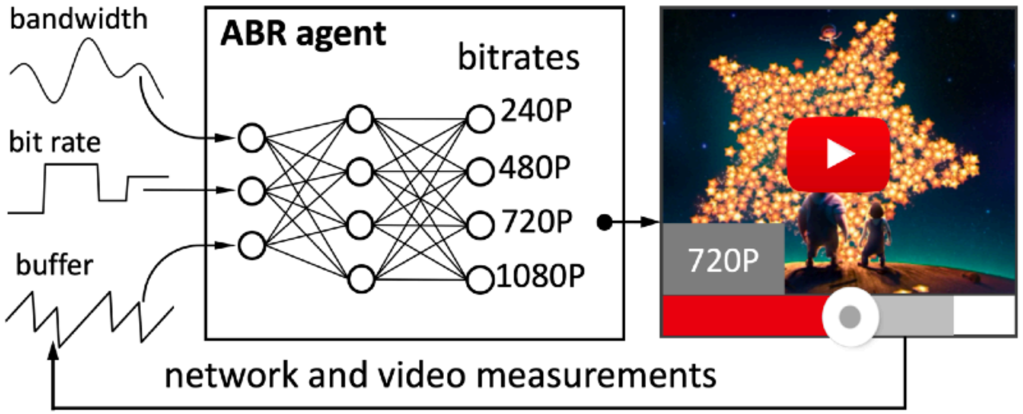
Hybrid: use both throughput prediction and buffer occupancy

- PBA (HotMobile'15), MPC (SIGCOMM'15)

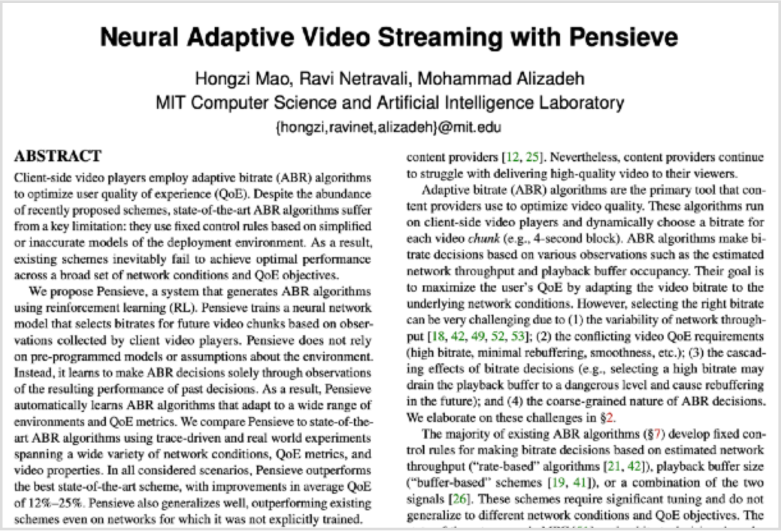
All these solutions are fixed heuristics and are based on the designer's insight. All of them rely on **simplified inaccurate model** which leads to suboptimal performance.

Can we automatically learn how to choose bitrates?

Pensieve: learning-based ABR algorithm



Pensieve learns ABR algorithm automatically through experience



Neural Adaptive Video Streaming with Pensieve

Hongzi Mao, Ravi Netravali, Mohammad Alizadeh
MIT Computer Science and Artificial Intelligence Laboratory
{hongzi,ravinet,alizadeh}@mit.edu

ABSTRACT

Client-side video players employ adaptive bitrate (ABR) algorithms to optimize user quality of experience (QoE). Despite the abundance of recently proposed schemes, state-of-the-art ABR algorithms suffer from a key limitation: they use fixed control rules based on simplified or inaccurate models of the deployment environment. As a result, existing schemes inevitably fail to achieve optimal performance across a broad set of network conditions and QoE objectives.

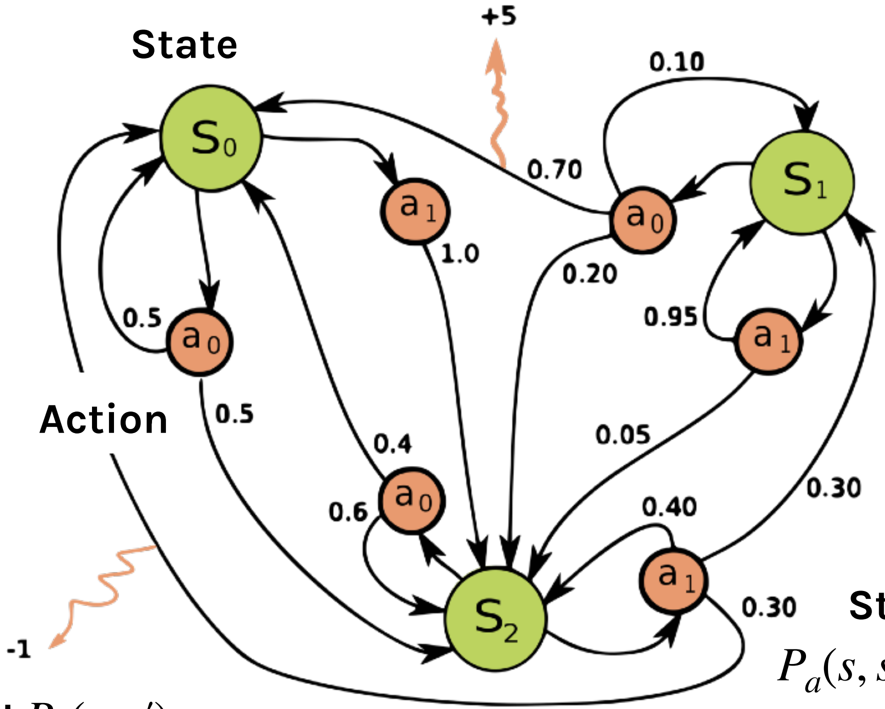
We propose Pensieve, a system that generates ABR algorithms using reinforcement learning (RL). Pensieve trains a neural network model that selects bitrates for future video chunks based on observations collected by client video players. Pensieve does not rely on pre-programmed models or assumptions about the environment. Instead, it learns to make ABR decisions solely through observations of the resulting performance of past decisions. As a result, Pensieve automatically learns ABR algorithms that adapt to a wide range of environments and QoE metrics. We compare Pensieve to state-of-the-art ABR algorithms using trace-driven and real world experiments spanning a wide variety of network conditions, QoE metrics, and video properties. In all considered scenarios, Pensieve outperforms the best state-of-the-art scheme, with improvements in average QoE of 12%–25%. Pensieve also generalizes well, outperforming existing schemes even on networks for which it was not explicitly trained.

content providers [12, 25]. Nevertheless, content providers continue to struggle with delivering high-quality video to their viewers.

Adaptive bitrate (ABR) algorithms are the primary tool that content providers use to optimize video quality. These algorithms run on client-side video players and dynamically choose a bitrate for each video *chunk* (e.g., 4-second block). ABR algorithms make bitrate decisions based on various observations such as the estimated network throughput and playback buffer occupancy. Their goal is to maximize the user's QoE by adapting the video bitrate to the underlying network conditions. However, selecting the right bitrate can be very challenging due to (1) the variability of network throughput [18, 42, 49, 52, 53]; (2) the conflicting video QoE requirements (high bitrate, minimal rebuffering, smoothness, etc.); (3) the cascading effects of bitrate decisions (e.g., selecting a high bitrate may drain the playback buffer to a dangerous level and cause rebuffering in the future); and (4) the coarse-grained nature of ABR decisions. We elaborate on these challenges in §2.

The majority of existing ABR algorithms (§7) develop fixed control rules for making bitrate decisions based on estimated network throughput ("rate-based" algorithms [21, 42]), playback buffer size ("buffer-based" schemes [19, 41]), or a combination of the two signals [26]. These schemes require significant tuning and do not generalize to different network conditions and QoE objectives. The

Markov decision process (MDP)



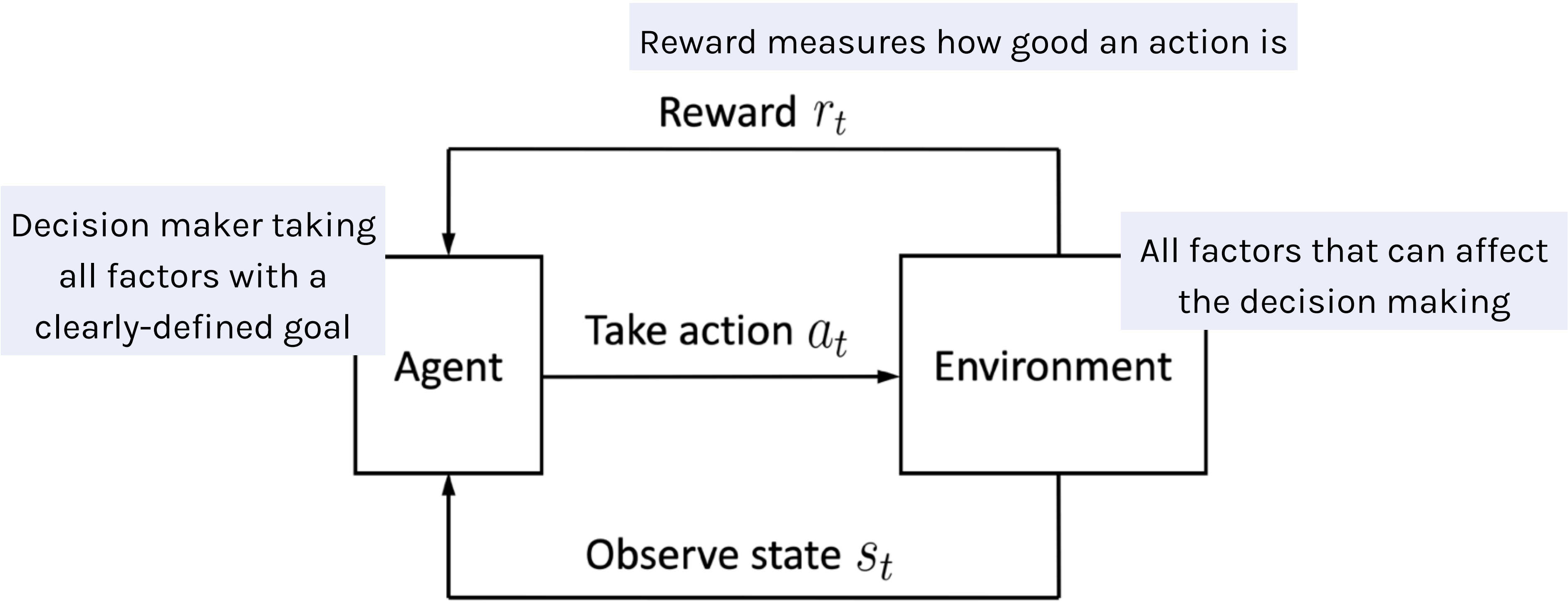
Reward $R_a(s, s')$

State transition probability

$$P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$$

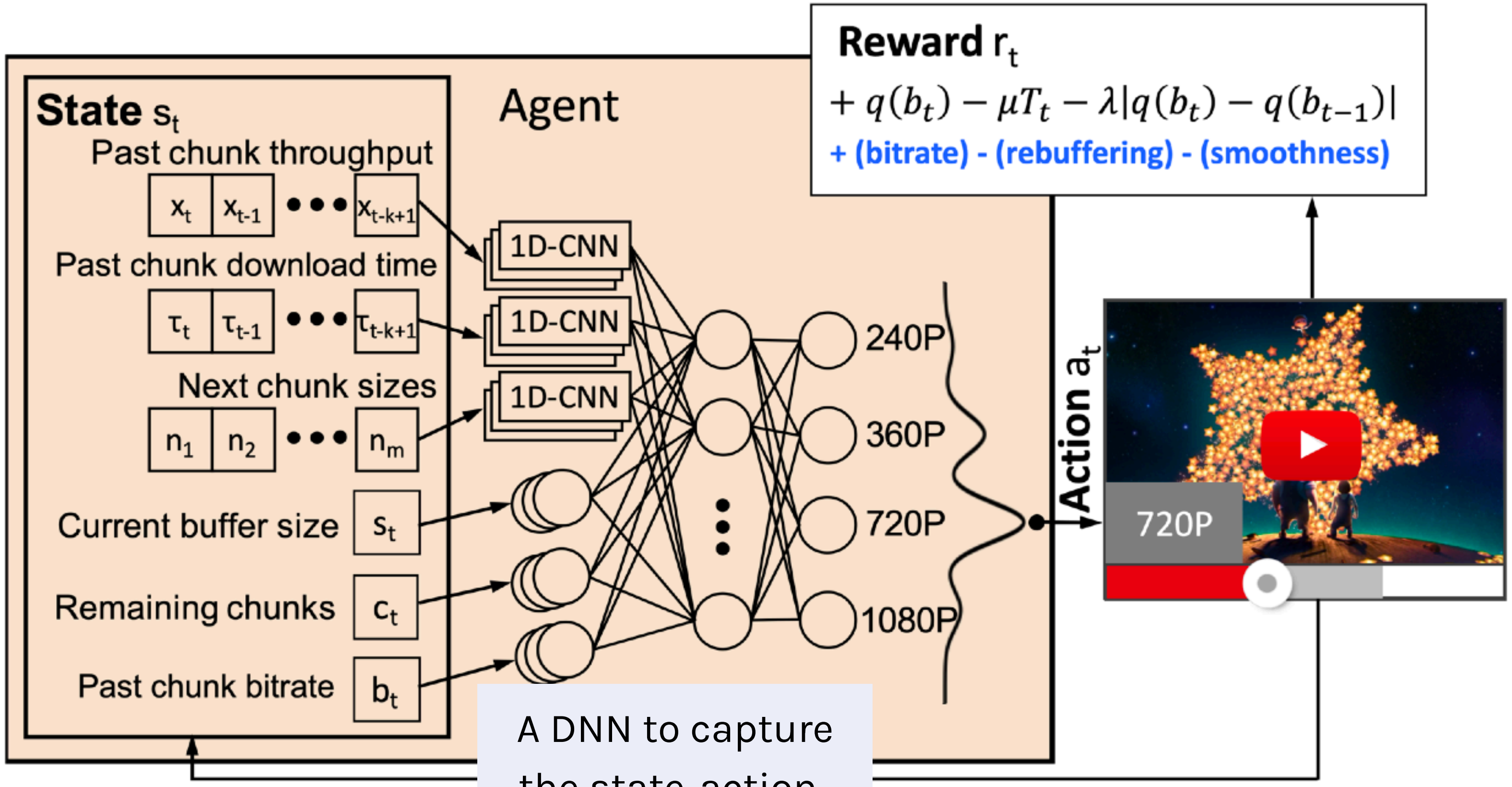
Markov chain: only one action for each state, all rewards are zero

Reinforcement learning



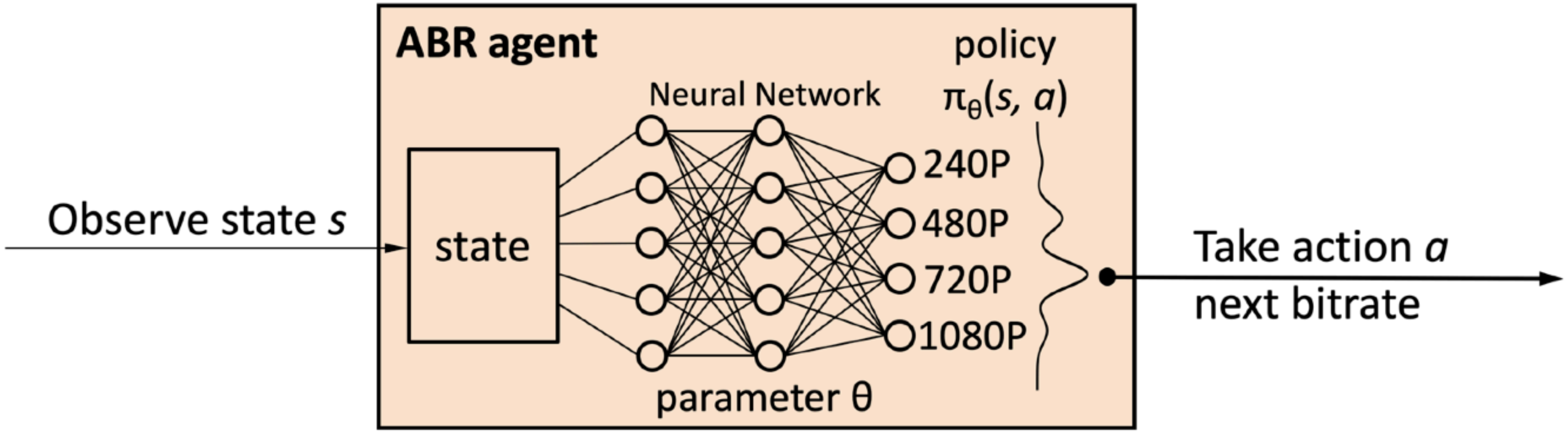
Goal: maximize the cumulative reward $\sum_t r_t$

Pensieve design



A DNN to capture the state-action-reward relationship

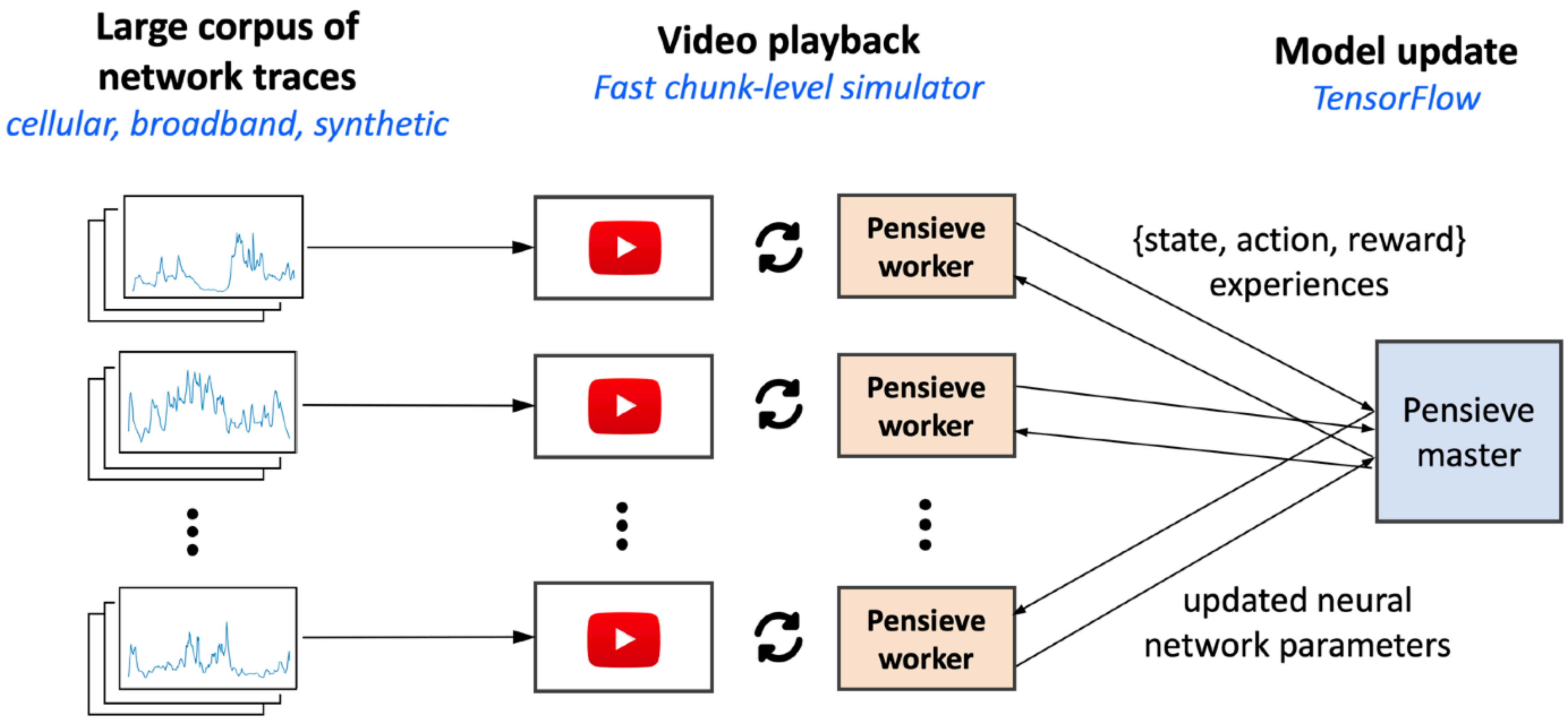
Training of the system



Train the system by **letting the system experience collected history data:** trajectories of [state, action, reward]

Gradient descent: $$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_t r_t\right]$$

Training of the system



Advantages of Pensieve

Learn the dynamics **directly from experience**

Optimize the high level QoE objective **end-to-end**

Extract control rules from **raw high-dimensional** signals

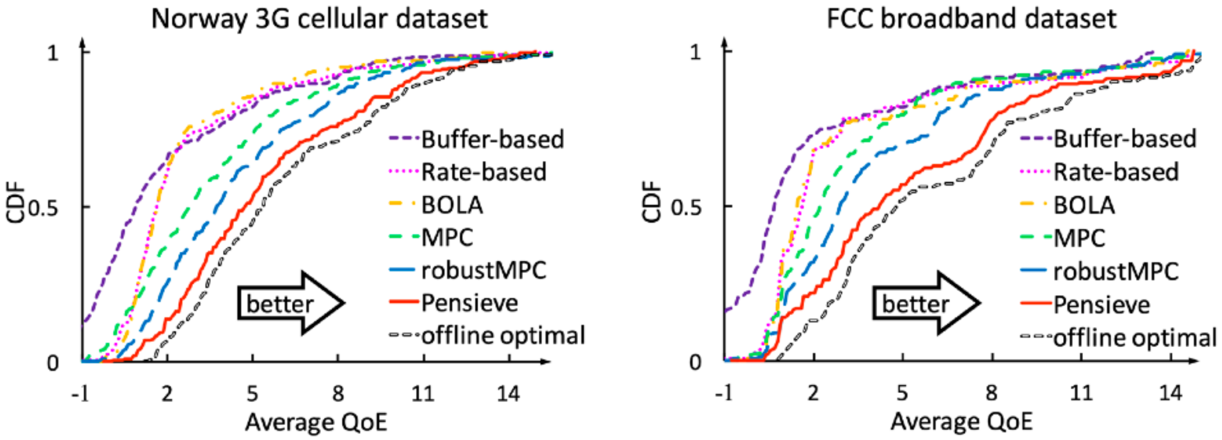
Trace-driven evaluation

Dataset

- Network traces: two datasets, each dataset consists of 1000 traces, each trace 320 seconds
- Video: 193 seconds, encoded at bitrates {300, 750, 1200, 2850, 4300} Kbps

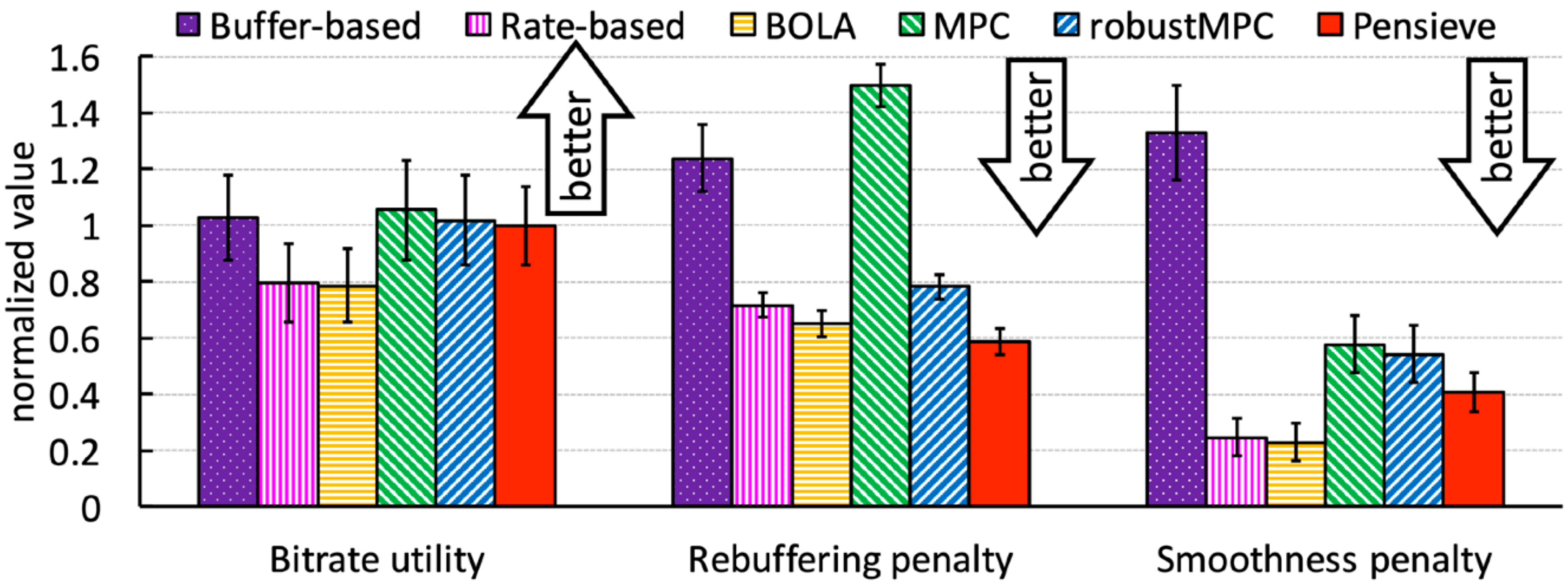
Video player and server

- Google Chrome browser with Apache web server



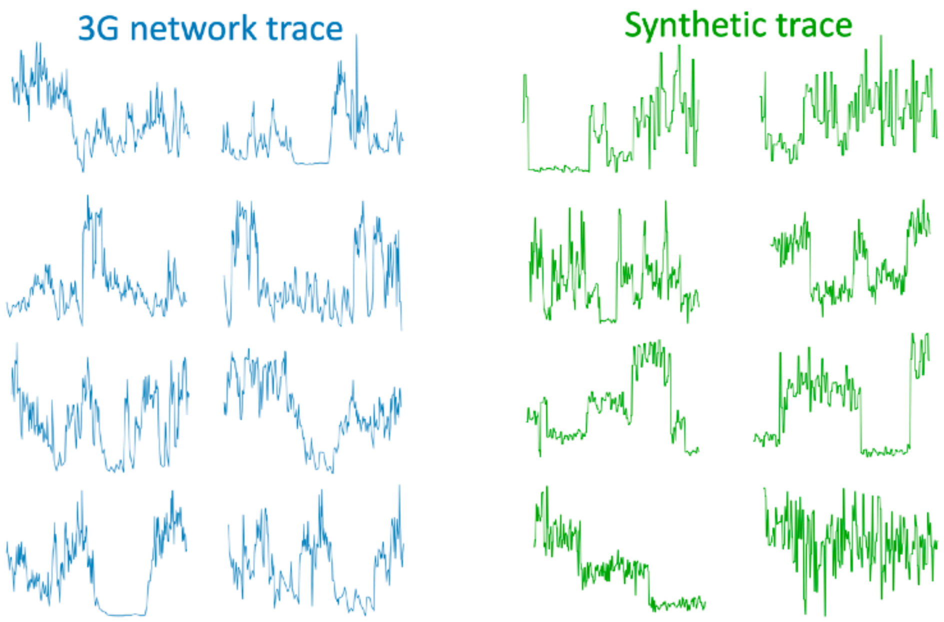
Pensieve improves the state-of-the-art by 12-25% and is within 9-14% of the offline optimal

QoE breakdown



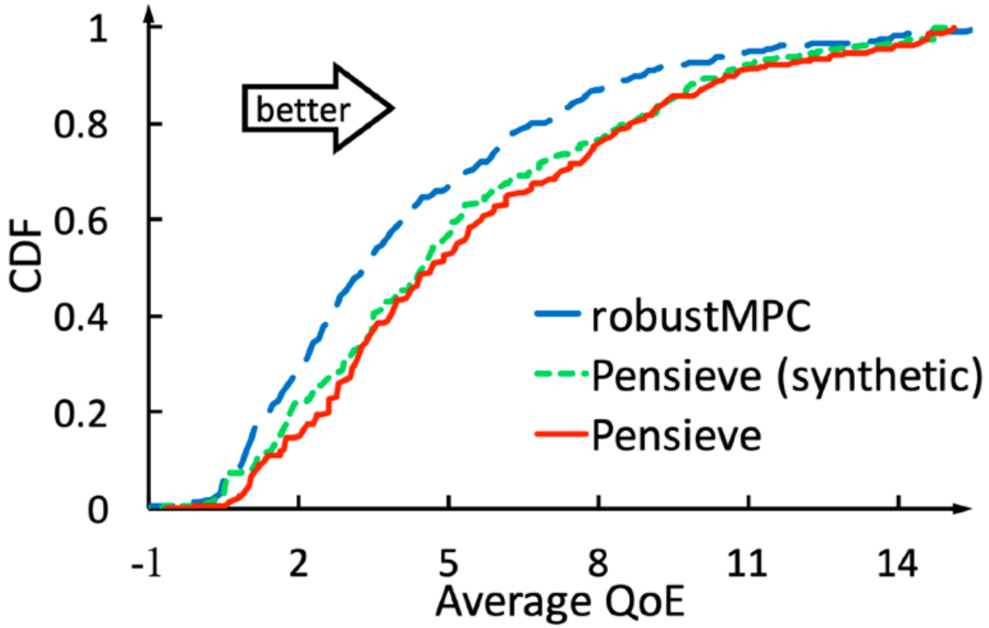
Pensieve reduces rebuffing by 10-32% over the state-of-the-art.

Generalization of Pensieve



Train Pensieve with synthetically generated (using a hidden Markov model) network traces, covering a wide range of average throughput and network variation.

Generalization of Pensieve



When we train Pensieve on synthetic network traces and test it on the real 3G network trace, we only see **~5% performance degradation**.

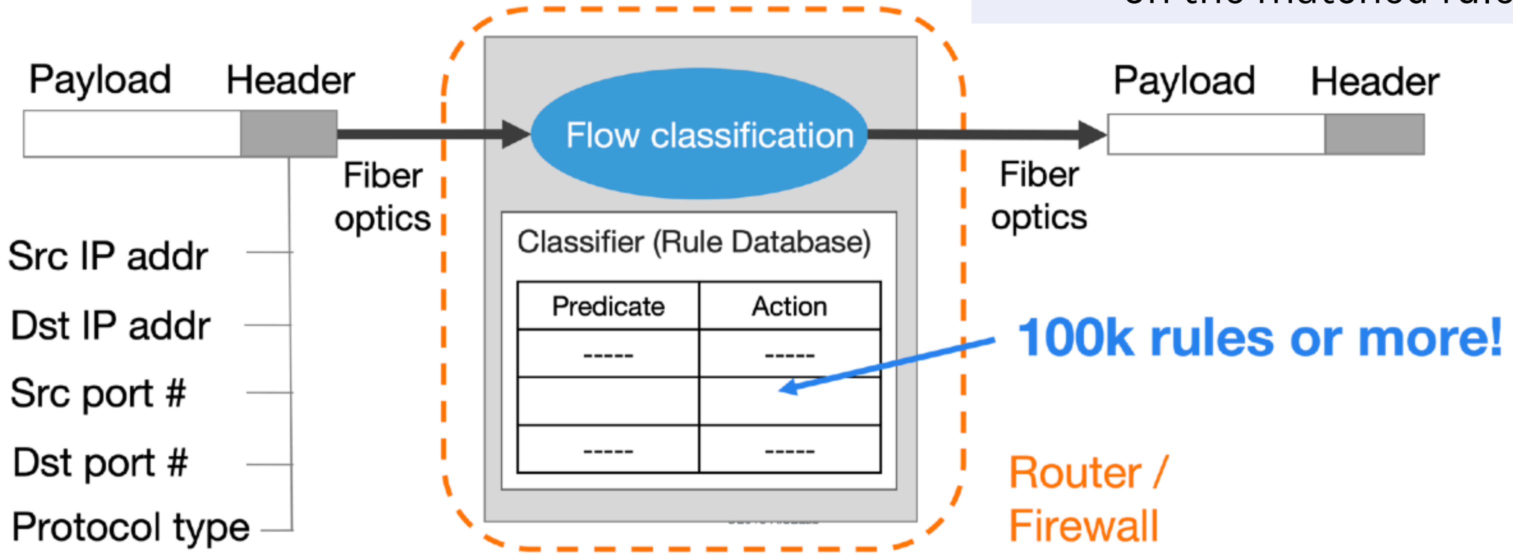
Machine Learning for Packet Classification

Packet classification

Fundamental problem in computer networking

- Building blocks for routing, access control, QoS, defense against attacks

Decides the action to take based on the matched rule



Packet classification example

Exact matching   Prefix matching   Range matching

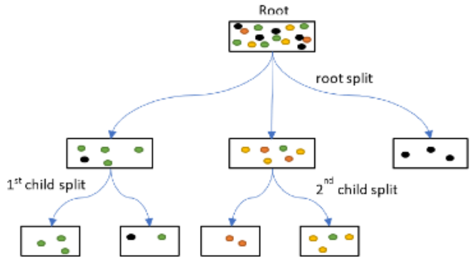
Priority	Src IP	Dst IP	Src Port	Dst Port	Protocol
2	10.0.0.0	10.0.0.0/16	*	*	*
1	*	*	[0, 1023]	[0, 1023]	TCP
0	*	*	*	*	*

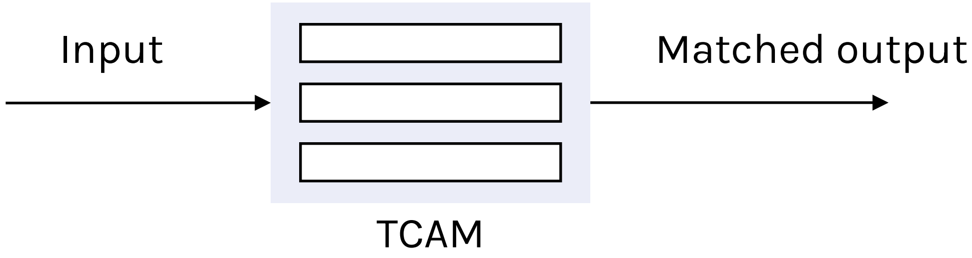
Example: **(10.0.0.0, 10.0.0.1, 0, 0, 6)**

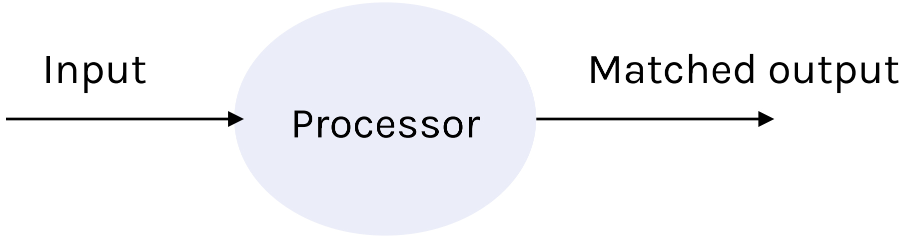
Any matching

Matches on all the 3 rules in the above table, but only the one with the highest priority will be taken.

Two approaches





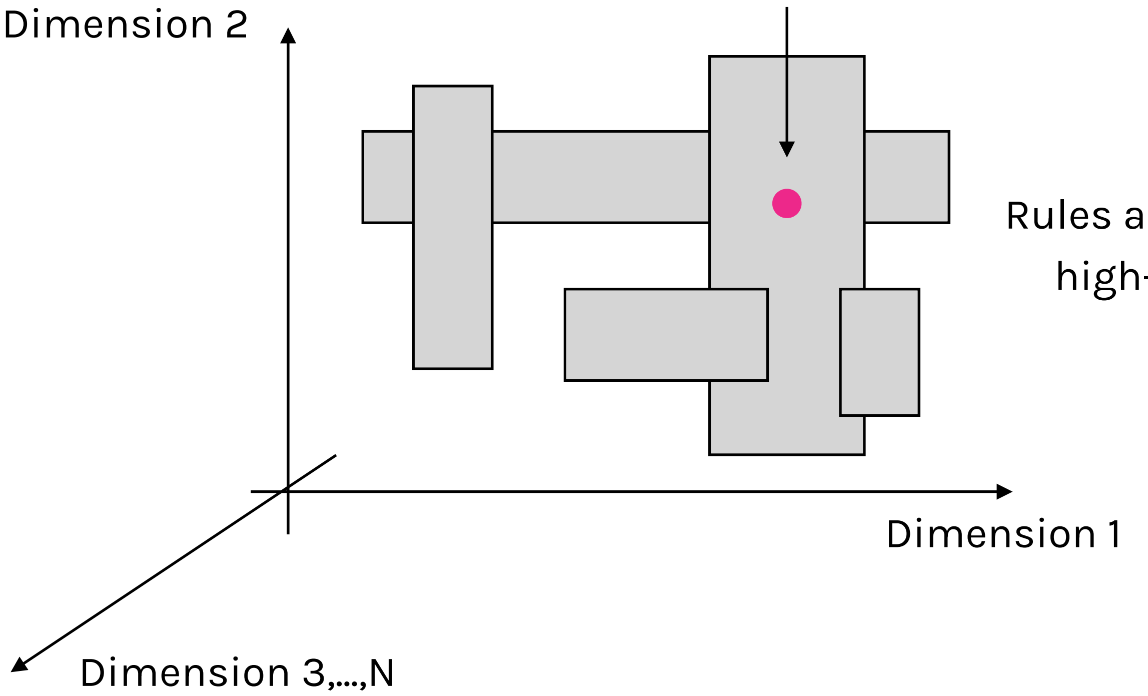


Hardware-based (e.g., TCAM): fast, expensive, energy-consuming, hard to scale

Software-based (e.g., decision-tree): scalable, slow and require large memory

Packet classification: a point-location problem

Problem: find out which hypercube contains the given point?



Rules are hypercubes in the high-dimension space

Theoretical analysis on packet classification complexity

Hard time-space tradeoff for point-location problem

- $O(\log N)$ time and $O(N^d)$ space
- $O(\log^d N)$ time and $O(N)$ space
- N: number of rules, d: number of attributes to match on, $N \approx 100K, d = 5$

TL;DR: logarithmic time, exponential space; linear space, exponential time → none of them is attractive

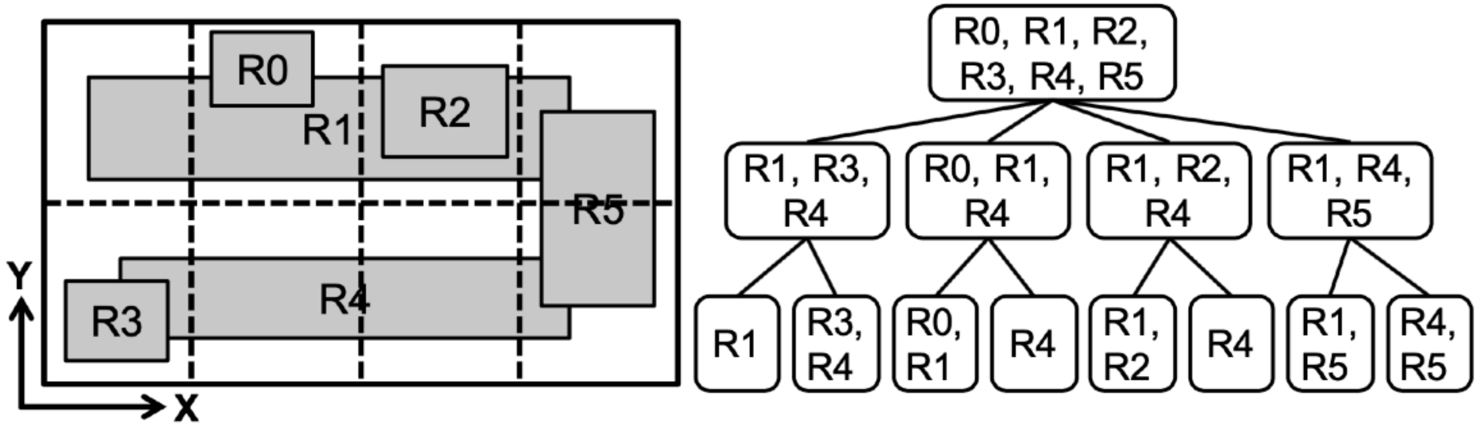
Even harder than the point-location problem

- Rules have priorities and can overlap with each other

http://yuba.stanford.edu/~nickm/papers/classification_tutorial_01.pdf

Existing techniques: node cutting

Cut the space into smaller areas and each area corresponds to a leaf in the decision tree



(a) Packet classifier.

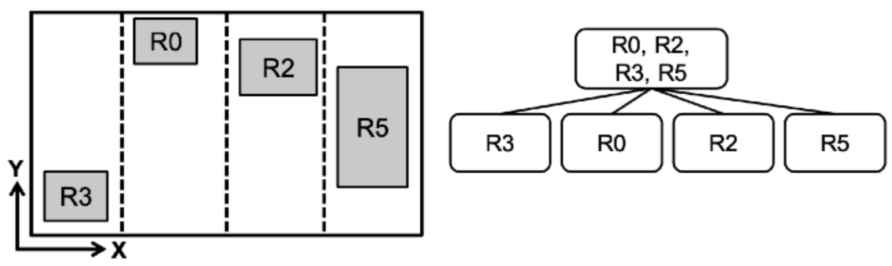
(b) Decision tree.

Match by traveling through the decision tree and select the rule in the matched leaf with the highest priority

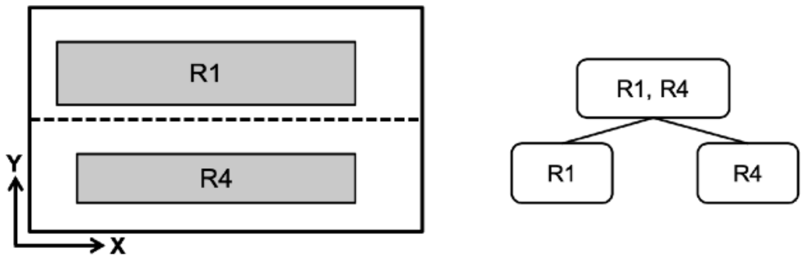
Fast but memory inefficient due to redundancies

Existing techniques: rule partitioning

Partition the space into two parts and build a separate decision tree (or a subtree) for each part



(a) Partition 1.

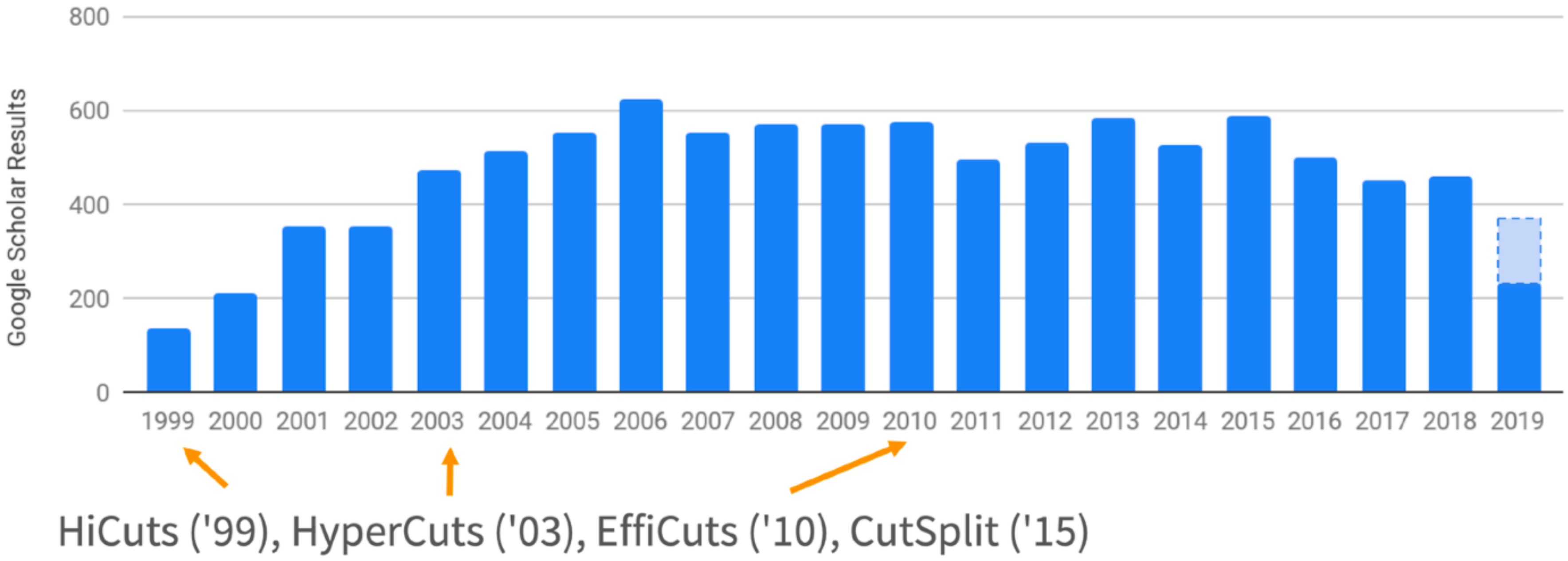


(b) Partition 2.

Match by traveling through all the decision tree and select the rule in the matched leaf in every tree with the highest priority

Memory efficient but slow due to the need to travel through all branches

20 years of active research



Almost all of these solutions are engineered with hand-tuned heuristics targeting different objectives

Can we apply learning?

Reinforcement learning **models long-term outcomes of actions** unlike heuristics

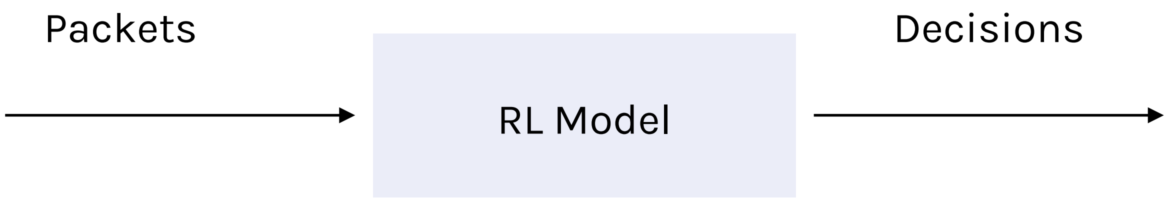
Reinforcement learning can **optimize for the end objectives** directly unlike heuristics

Historically, efficient RL formulation means super-human performance (e.g., AlphaZero, AlphaStar, AlphaFold).

End-to-end learning

Replace the decision tree with a DNN or an RL agent, does this work?

End-to-end learning



Pros

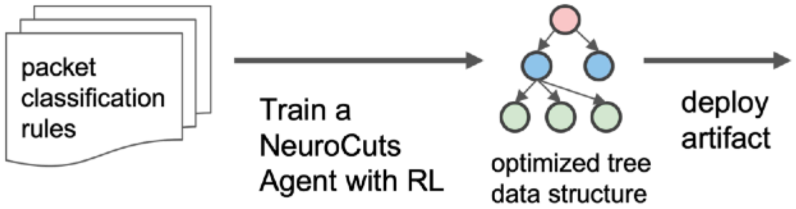
- May not need to build a data structure at all

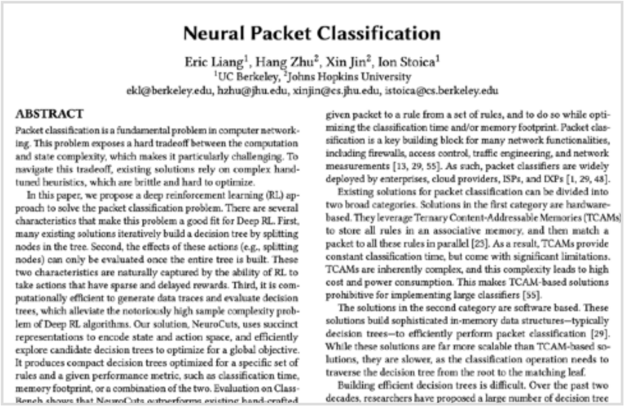
Cons

- **Cannot guarantee classification correctness** (critical for applications like access control)
- Very large space of inputs → **hard to check model correctness**
- Packet inference **takes too long** (required time within 100s of ns)
- Need **specialized inference hardware** (e.g., GPU, TPU)

NeuroCuts

Use deep reinforcement learning to tackle the problem of building decision trees, instead of applying per-packet inference directly





Neural Packet Classification

Eric Liang[1], Hang Zhu[2], Xin Jin[2], Ion Stoica[1]
[1]UC Berkeley, [2]Johns Hopkins University
ekl@berkeley.edu, hzhu@jhu.edu, xinjin@cs.jhu.edu, istoica@cs.berkeley.edu

ABSTRACT

Packet classification is a fundamental problem in computer networking. This problem exposes a hard tradeoff between the computation and state complexity, which makes it particularly challenging. To navigate this tradeoff, existing solutions rely on complex hand-tuned heuristics, which are brittle and hard to optimize.

In this paper, we propose a deep reinforcement learning (RL) approach to solve the packet classification problem. There are several characteristics that make this problem a good fit for Deep RL. First, many existing solutions iteratively build a decision tree by splitting nodes in the tree. Second, the effects of these actions (e.g., splitting nodes) can only be evaluated once the entire tree is built. These two characteristics are naturally captured by the ability of RL to take actions that have sparse and delayed rewards. Third, it is computationally efficient to generate data traces and evaluate decision trees, which alleviate the notoriously high sample complexity problem of Deep RL algorithms. Our solution, NeuroCuts, uses succinct representations to encode state and action space, and efficiently explore candidate decision trees to optimize for a global objective. It produces compact decision trees optimized for a specific set of rules and a given performance metric, such as classification time, memory footprint, or a combination of the two. Evaluation on ClassBench shows that NeuroCuts outperforms existing hand-crafted

given packet to a rule from a set of rules, and to do so while optimizing the classification time and/or memory footprint. Packet classification is a key building block for many network functionalities, including firewalls, access control, traffic engineering, and network measurements [13, 29, 55]. As such, packet classifiers are widely deployed by enterprises, cloud providers, ISPs, and IXPs [1, 29, 48].

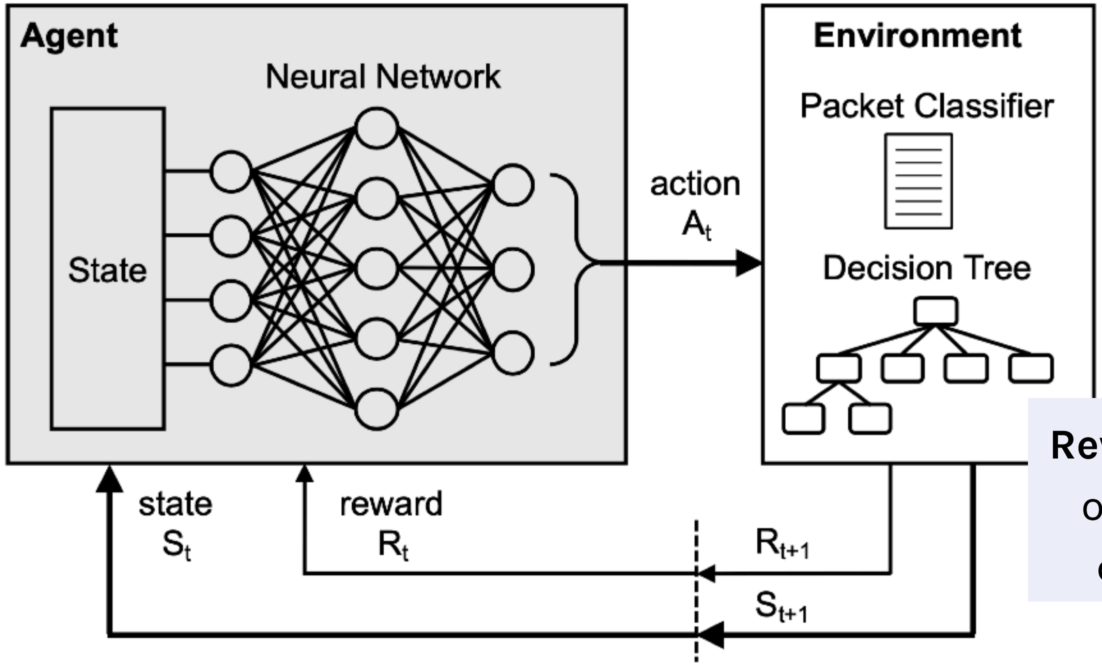
Existing solutions for packet classification can be divided into two broad categories. Solutions in the first category are hardware-based. They leverage Ternary Content-Addressable Memories (TCAMs) to store all rules in an associative memory, and then match a packet to all these rules in parallel [23]. As a result, TCAMs provide constant classification time, but come with significant limitations. TCAMs are inherently complex, and this complexity leads to high cost and power consumption. This makes TCAM-based solutions prohibitive for implementing large classifiers [55].

The solutions in the second category are software based. These solutions build sophisticated in-memory data structures—typically decision trees—to efficiently perform packet classification [29]. While these solutions are far more scalable than TCAM-based solutions, they are slower, as the classification operation needs to traverse the decision tree from the root to the matching leaf.

Building efficient decision trees is difficult. Over the past two decades, researchers have proposed a large number of decision tree

NeuroCuts design

Action: either cutting a node or partitioning a set of rules



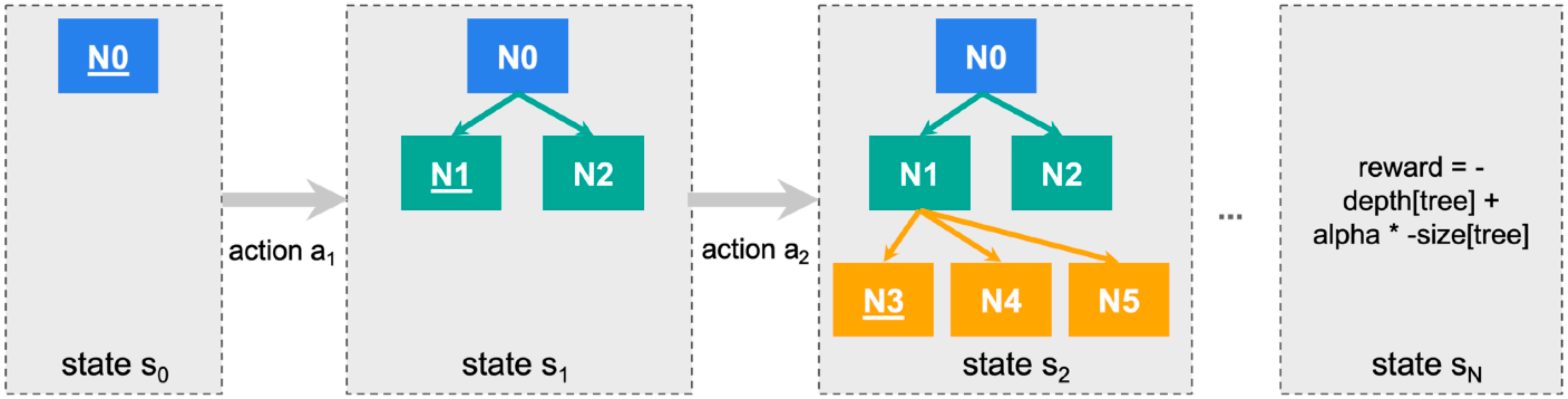
Reward: classification time, or memory footprint, or a combination of the two

The reward is delayed and is only given when the whole tree is built.

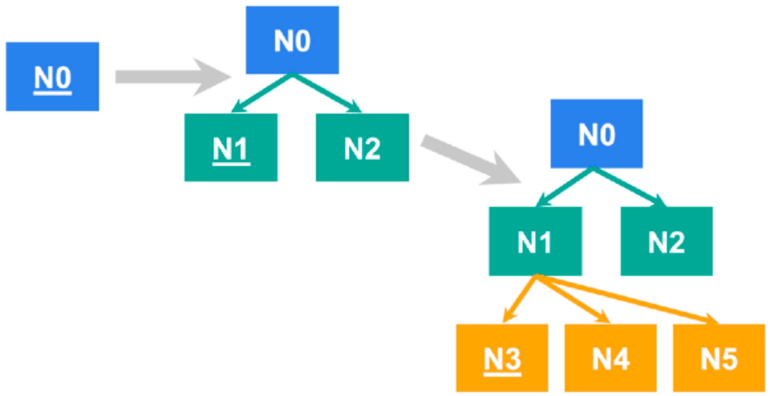
Naive MDP formulation

Sequential Markov Decision Process (MDP)

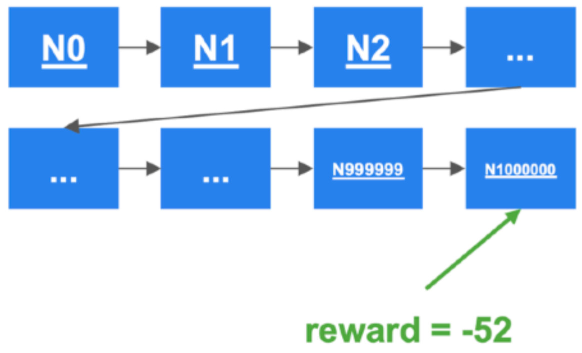
- Assumes Depth-First Search (DFS) order of building the tree node by node
- Action is to **cut** or **partition** current node



Challenges



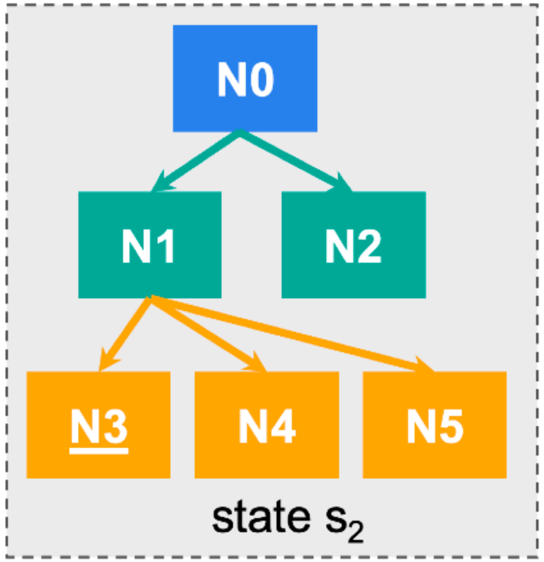
Size of the state grows in each step:
hard to define the state



Reward delayed until the end:
sparse reward problem

Challenge 1: state definition

Observation: node state is independent from the parent and sibling nodes



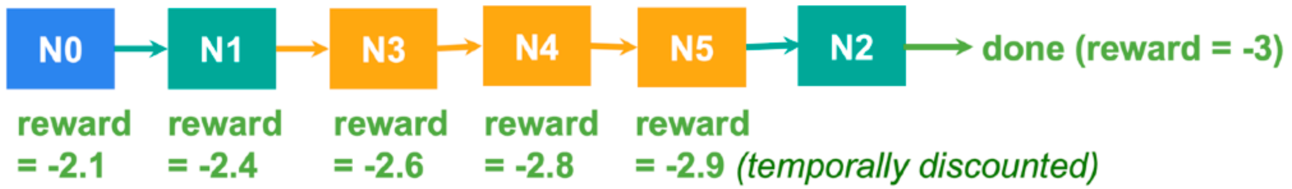
s2 can be represented as a fixed-length vector describing N3's bounding hypercube

```
{SrcIPMin, SrcIPMax,
 DstIPMin, DstIPMax,
 SrcPortMin, SrcPortMax,
 DstPortMin, DstPortMax,
 ProtocolMin, ProtocolMax}
```

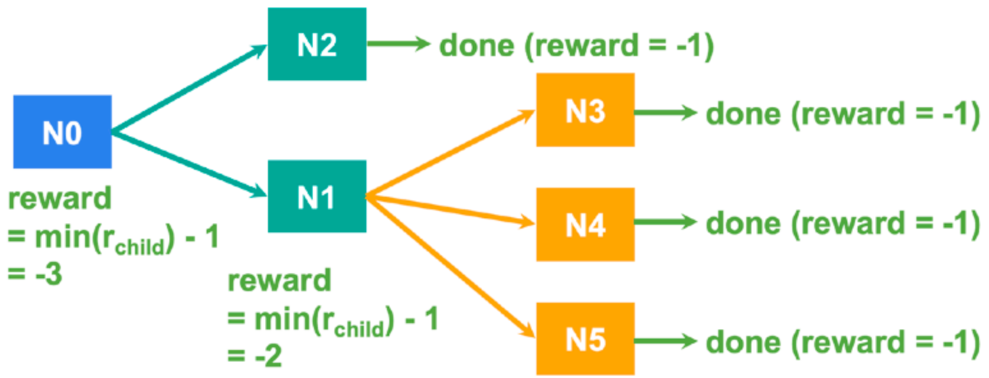
Challenge 2: reward

Observation: building a tree is a branching decision process, not sequential MDP

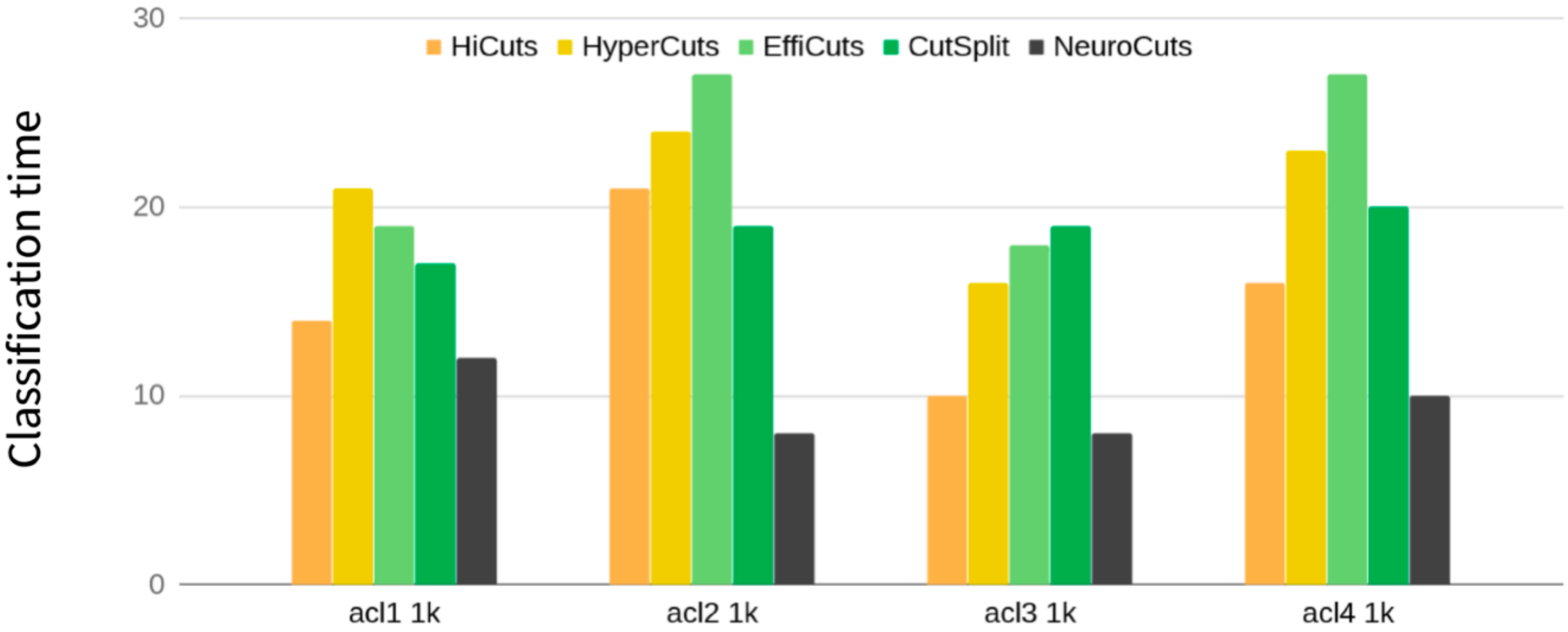
Sequential MDP: O(n) steps delay between action time and reward time



Branching decision process: O(logn) steps delay between action time and reward time

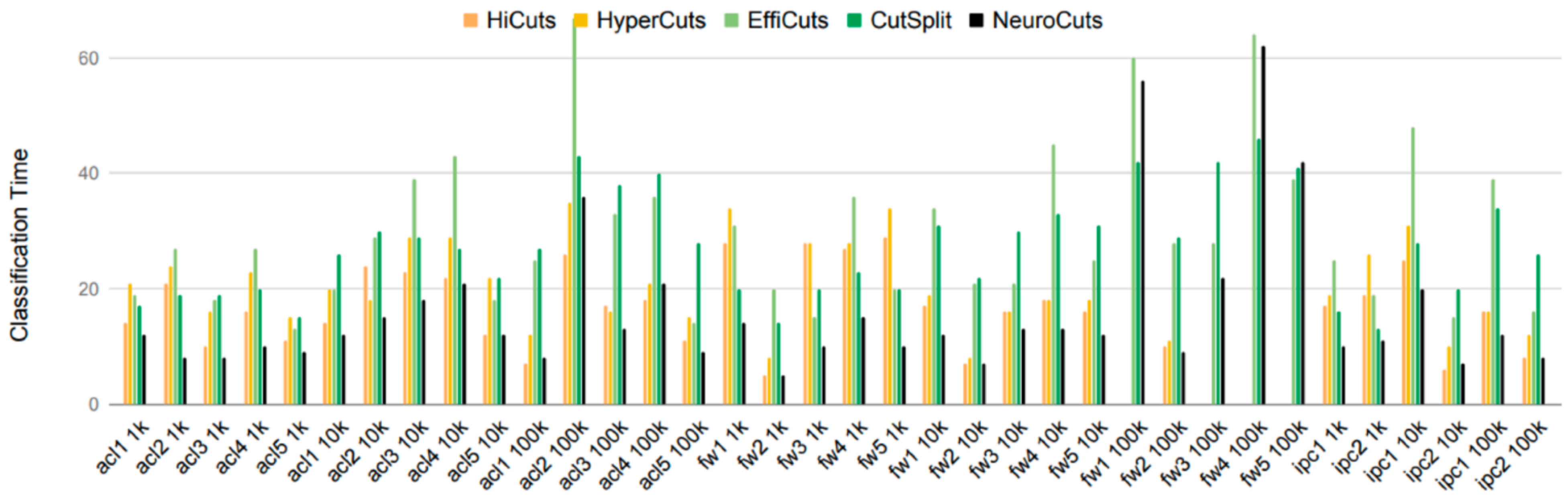


Result: classification time



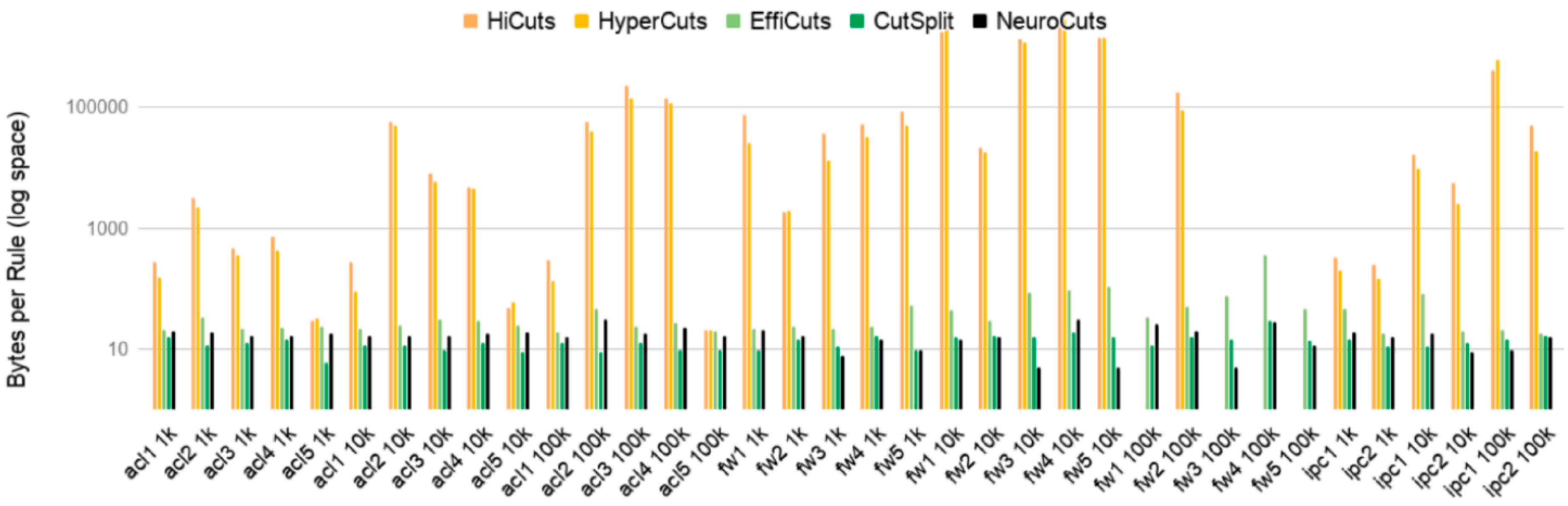
NeuroCuts significantly improves the classification time over the state-of-the-art heuristic approaches.

Results: scalability



NeuroCuts scales to large rule sets and achieves 18% (median) time improvement (up to 2x).

Result: space efficiency



Up to 3x better memory over all baselines. CutSplit is better at median.

Machine learning for other networking problems

Network routing

- Deciding how packets should be forwarded on a network by learning
- Optimizing network utilization, congestion, etc.

Congestion control

- Deciding how to control the congestion window by learning
- Accounting for multiple objectives: throughput, latency, smoothness

Cache management

- Learning-based CDN cache eviction policies

Machine Learning for Computer Systems and Networking: A Survey

MARIOS EVANGELOS KANAKIS, Vrije Universiteit Amsterdam
RAMIN KHALILI, Huawei Munich Research Center
LIN WANG, Vrije Universiteit Amsterdam and TU Darmstadt

Machine learning (ML) has become the de-facto approach for various scientific domains such as computer vision and natural language processing. Despite recent breakthroughs, machine learning has only made its way into the fundamental challenges in computer systems and networking recently. This article attempts to shed light on recent literature that appeals for machine learning-based solutions to traditional problems in computer systems and networking. To this end, we first introduce a taxonomy based on a set of major research problem domains. Then, we present a comprehensive review per domain, where we compare the traditional approaches against the machine learning-based ones. Finally, we discuss the general limitations of machine learning for computer systems and networking, including lack of training data, training overhead, real-time performance, and explainability, and reveal future research directions targeting these limitations.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Computer systems organization**; • **Networks**;

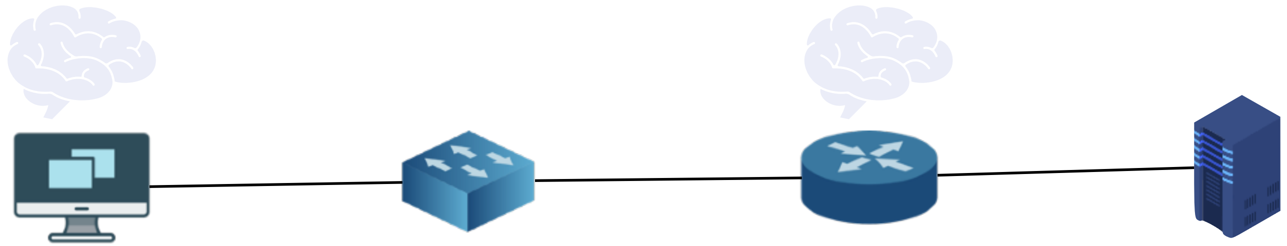
Additional Key Words and Phrases: Machine learning, computer systems, computer networking

ACM Reference format:
Marios Evangelos Kanakis, Ramin Khalili, and Lin Wang. 2022. Machine Learning for Computer Systems and Networking: A Survey. *ACM Comput. Surv.* 55, 4, Article 71 (November 2022), 36 pages.
https://doi.org/10.1145/3523057

1 INTRODUCTION

Revolutionary research in **machine learning (ML)** has significantly disrupted the scientific community by contributing solutions to long-lived challenges. Thanks to the continuous advancements in computing resources (e.g., cloud data centers) and performance capabilities of processing units (e.g., accelerators like GPUs and TPUs), ML, particularly its rather computation-expensive subset namely **deep learning (DL)**, has gained its traction [120, 131]. In general, ML has established dominance in vision tasks such as image classification, object recognition [86], and more to follow [58, 156]. Other remarkable examples where ML is thriving include speech recognition [52]

Summary



Machine learning can be leverage to solve the decision-making problems in networking, e.g., adaptive bitrate selection and packet classification.

Next time: course summary