Explaining Deep Learning-Based Networked Systems

Zili Meng
Tsinghua University
zilim@ieee.org

Minhu Wang
Tsinghua University
wangmh19@mails.tsinghua.edu.cn

Mingwei Xu
Tsinghua University
xumw@tsinghua.edu.cn

Hongzi Mao
Massachusetts Institute of Technology
hongzi@mit.edu

Jiasong Bai
Tsinghua University
bjs17@mails.tsinghua.edu.cn

Hongxin Hu
Clemson University
hongxih@clemson.edu

Abstract

While deep learning (DL)-based networked systems have shown great potential in various applications, a key drawback is that Deep Neural Networks (DNNs) in DL are blackboxes and *nontransparent* for network operators. The lack of interpretability makes DL-based networked systems challenging to operate and troubleshoot, which further prevents DL-based networked systems from deploying in practice. In this paper, we propose TranSys, a novel framework to explain DL-based networked systems for practical deployment. TranSys categorizes current DL-based networked systems and introduces different explanation methods based on decision tree and hypergraph to effectively explain DL-based networked systems. TranSys can explain the DNN policies in the form of decision trees and highlight critical components based on analysis over hypergraph. We evaluate TranSys over several typical DL-based networked systems and demonstrate that TranSys can provide human-readable explanations for network operators. We also present three use cases of TranSys, which could (*i*) help network operators troubleshoot DL-based networked systems, (*ii*) improve the decision latency and resource consumption of DL-based networked systems by $\sim$10$\times$ on different metrics, and (*iii*) provide suggestions on daily operations for network operators when incidences occur.

1 Introduction

Machine learning, especially deep learning, has emerged as a general and powerful approach to many complex optimization problems. In the networking community, many recent studies have adopted deep learning (DL)-based approaches for video streaming [53, 76], local traffic control [18, 40], parallel job scheduling [45, 54, 55], and network optimization [64, 75]. Among all machine learning methods, Deep Neural Networks (DNNs) have strong prediction ability and standardized optimization procedure [58], and have been accordingly adopted to improve the performance of networked systems [18, 53, 54] compared to human-crafted heuristics.

However, the promising performance of DNNs also comes with significant challenges. Since DNNs can easily have thousands or even millions of neurons, understanding the decision logic of DNNs is difficult. Network operators have to consider those DNNs as large *blackboxes*. This could result in severe issues in system troubleshooting, lightweight deployment, and daily operation: (*i*) when DL-based networked systems make sub-optimal decisions, network operators have no clue on how to fix the problem; (*ii*) when systems are going to be deployed onto network devices, the resource consumption and decision latency are much worse than those of heuristics; and (*iii*) when operators want to perform additional manual operation actions (e.g., rerouting when congestion occurs), they have to retrain the learned model or test the actions in practice, which usually takes considerable time and expenses. The drawbacks above result in a general fear against DNNs for network operators [26, 81] and prevent DNNs from deployment in the real world.

Fortunately, the machine learning community has been recently active on proposing interpretable learning methods, including (super)linear functions [48, 61], decision trees [12], finite state machines (FSMs) [44], etc. However, many methods are pinned at specific scenarios and have their performance issues. For example, the recently proposed FSM-based method has drastic performance loss even when scaled to simple Atari games such as Breakout[1] [44]. For heterogeneous and sophisticated networked systems, which sometimes take much more states and high-order structural information (e.g., network topology) as input, introducing a general framework to provide explanations is challenging.

To make the best use of a wealth of these approaches, we categorize the networked systems into two main groups. First, many networked systems do not have a global view of the network and make decisions only based on their local observations. We categorize them into local control systems (LCSes). For example, Pensieve [53] adjusts the bitrate of video streaming based on the estimated throughput and

[1]The state space of Breakout has eight dimensions only.

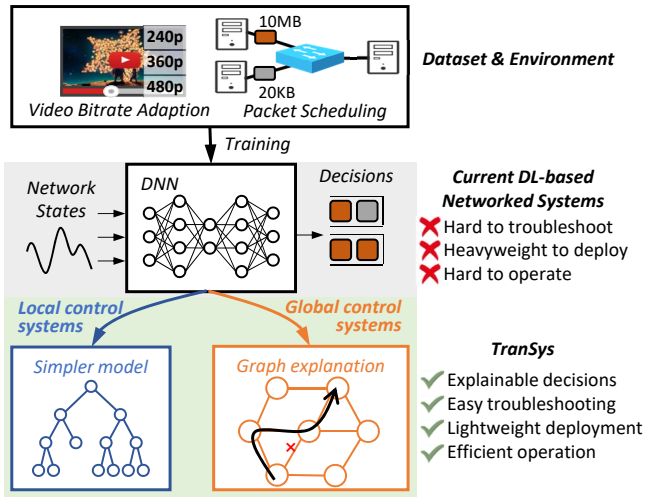


Figure 1: Current DL-based systems *v.s.* TranSys.

buffer occupancy on end devices. Since the information that could be collected by LCSes are usually not very much, the state space of those LCSes will not be huge. The small state space enables us to convert the DNNs in LCSes into simpler models based on current approaches with negligible performance loss. Second, for those networked systems that have a global view of the overall situations, most of them take the *graph-based* information (e.g., topology) as input. We categorize them into global control systems (GCSes). The wide adoption of the graph is mainly because the network itself is a graph. For example, a traffic optimizer on the control plane takes the topology and all the data plane devices as input and optimizes performance goals such as flow completion time [64]. The graph essence of GCSes enables us to unify them and provide explanations with designs over graphs.

Thus, we propose TranSys[2], a novel framework to explain DL-based networked systems for practical deployment. As shown in Figure 1, we first categorize current networked systems into LCSes and GCSes based on their observation horizons [§3]. For LCSes, we convert DNNs into *decision trees* to make their internal decision process interpretable. We choose decision tree as the target model with the intuition that the structure of decision tree is similar to the decision logic of networked systems [§4.1]. We adopt a reinforcement learning verification method [12] to reduce the performance loss during the conversion. For GCSes, we propose a unified mathematical model based on *hypergraph*, which could be applied to many GCSes. We then explain the results generated from hypergraphs by finding the critical components that have significant influences on the outputs. With TranSys, network operators could enjoy the performance benefit brought by DL and also have an interpretable decision process. TranSys could further bring several benefits from deployment to operation [§7].

Results highlights. We apply TranSys over several DL-based networked systems and explain how they make decisions. For example, we explain the policy of a DL-based adaptive bitrate (ABR) system [53] and find a new important decision variable. We also explain a DL-based traffic optimizer and find out the significant components (e.g., critical branches) that affect the results. Furthermore, we present three use cases of TranSys on current DL-based networked systems. (*i*) We troubleshoot the DNN in the DL-based ABR system and improve the average quality of experience (QoE) by up to 3% over DNN policies with only decision trees [§7.1]. (*ii*) With decision trees generated by TranSys, we lightweightify two DL-based networked systems and achieve shorter decision-making latency (by 27× on average) and lower resource consumption (by up to 156×) [§7.2]. (*iii*) We also provide an efficient way to compare the latency of several paths in traffic optimization based on the explanations provided by TranSys [§7.3].

Table 1: Two categories of DL-based networked systems.

Category	Scenario	Examples
LCS	End-based congestion control	Aurora [40]
	Client-based video streaming	Pensieve [53]
	On-switch flow scheduling	AuTO [18]
GCS	Cluster job scheduling	Decima [54]
	Global network optimization	RouteNet [64]
	Network function placement	NFVdeep [75]

Contributions. We make the following contributions:

- We propose TranSys, a novel framework to explain DL-based networked systems, and its two-part designs for LCSes and GCSes, respectively [§3].
- For LCSes, we explain their behaviors by faithfully converting their DNNs into decision trees to clearly understand their internal decision logic [§4].
- For GCSes, we propose a unified hypergraph model for networked systems and a novel explanation method by finding critical elements in hypergraphs [§5].
- Evaluation with three DL-based networked systems shows that TranSys could provide human-readable explanations [§6]. We also present multiple use cases from troubleshooting to lightweight deployment [§7].

To the best of our knowledge, TranSys is the first framework to enable the explanation of diverse DL-based networked systems with respect to practical deployment. TranSys is available at `https://transys.io`. We believe that TranSys will accelerate the deployment of DL-based networked systems in practice.

2 Background and Motivations

2.1 DL-based Networked Systems

We first categorize current DL-based networked systems into local control systems (LCSes) and global control systems (GCSes) based on their observation horizon. We present several examples in both categories in Table 1.

[2] TranSys stands for **trans**parent networked **sys**tems.

LCSes are networked systems that make decisions based on their local (or end) information. For example, the congestion control agent adjusts the congestion window based on its local network observations from the end device (e.g., round-trip time or packet drop) [40]. Adaptive video streaming algorithms select the optimal bitrate to download according to the buffer occupancy and bandwidth prediction [53].

GCSes have the global information (e.g., from the view of the control plane) and take both the observations from each node and also the structural graph information as input. For example, a Software Defined Networking (SDN) network optimizer makes routing decisions based on the observations from all data plane switches and *also the network topology* [64]. A cluster job scheduler in Spark takes those *graph-based jobs* as input and decides how to allocate the resources to jobs [54]. Since GCSes need to process the structural graph information (e.g., network topology) of networked systems [§5], most GCSes employ the design of Graph Neural Network (GNN) [74]. GNN is an embedding method to capture the dependencies between the nodes of graphs. Appendix A.1 provides a brief primer on GNN.

Meanwhile, networked systems have a similar essence to the sequential decision-making process. For networked systems, the decision agent continuously collects network measurement statistics and makes decisions on flow scheduling, video bitrate selection, *etc.*, with their specific optimization goals. Moreover, the decisions made at the current step will also affect the future observations of the agent, which makes the adoption of machine learning challenging.

2.2 Motivations

Although DNNs improves the performance of networked systems, their blackbox property leads to several drawbacks for network operators, which motivate the design of TranSys.

(*i*) **Difficult to troubleshoot.** DNNs could contain thousands or even millions of neurons. When the DL-based networked systems fail to achieve the expected performance, network operators will have difficulty in locating the erroneous component due to the complexity of DNN. Moreover, different from other machine learning applications, problems in networked systems usually have easily adjustable solutions. For example, we could adjust the weights for different jobs in fair scheduling to catch up with the variations in workloads [54]. Nonetheless, troubleshooting networked systems has already been challenging for network operators for decades. Introducing DNNs will make the matter worse.

(*ii*) **Too heavyweight to deploy.** DNNs are known to have high resource consumption and long decision latency [20]. In contrast, networked systems usually require ultra-low latency and resource consumption to ensure high processing performance. These requirements are especially important for LCSes since they are deployed on end devices (e.g., mobile phones) or in-network devices (e.g., switches), which

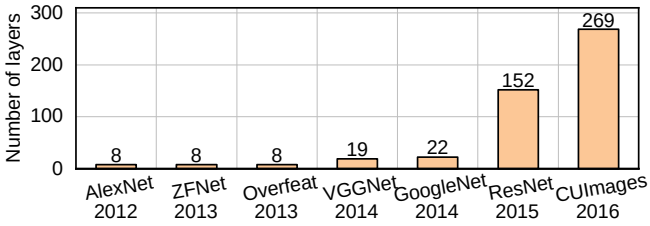


Figure 2: The sharp increase of DNN complexity in ImageNet Challenge winners [25, 36] (adopted from [28]).

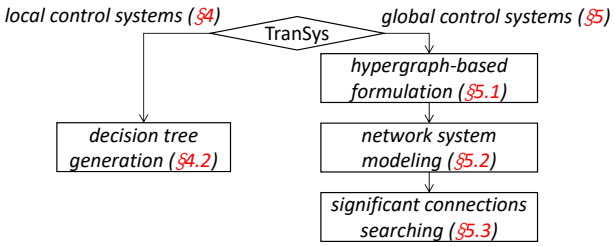


Figure 3: TranSys workflow.

are usually resource-limited and latency-sensitive [39]. For example, initializing a DNN-based ABR algorithm on clients increases the page load time by ∼10 seconds [§7.2.2], which will make users leave the page [27].

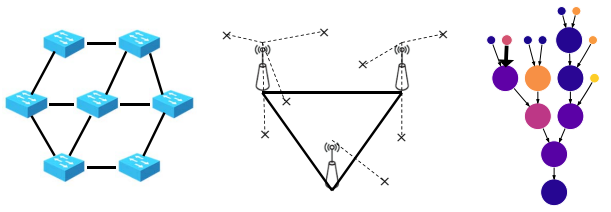
(*iii*) **Difficult to operate.** Although the internal structure of DNN is known to network operators, due to the high complexity of DNN, it is hard to understand how DNNs make decisions. The lack of causality will bring difficulties to network operators when they need to operate, upgrade, or predict the networked systems. For example, when the end-to-end latency of a certain path suddenly increases, without explanations of decisions, network operators might not know which paths should be rerouted among many candidate paths (e.g., in a fat-tree topology).

Note that the application of DNNs in DL-based networked systems is still at a preliminary stage: Pensieve (designed in 2017) has only one hidden layer, and AuTO (designed in 2018) has less than ten hidden layers. As a comparison, to make neural networks easier to train [24], a sharp increase of the number of neural network layers (hundreds or thousands of layers [36]) has been observed in other communities (Figure 2). Although we are not saying that the deeper is the better, it is indisputable that deeper neural networks will further aggravate the problem.

In conclusion, although DL-based networked systems have impressive performance improvements, without addressing the drawbacks above, it would be hard to deploy the DL-based systems in practice. This motivates us to explain and then practicalize DL-based networked systems.

3 Design Overview

The workflow of TranSys is presented in Figure 3. As introduced in §1, TranSys first categorizes DL-based networked systems into LCSes and GCSes, and explains them with



a) Wired network b) Cellular network c) Spark query

Figure 4: Many GCS scenarios are composed with graphs.

different methods. There are two observations behind the adoption of two different explanation methods over two categories of systems: (*i*) For LCSes, due to their simplicity, we could convert the DNNs into simpler models without the loss of performance; (*ii*) For GCSes, even if they are more complicated, the graph-based essence of them enables us to provide targeted explanations with analysis over graphs. We further discuss our adoption of two explanation methods and their differences in §8.

For LCSes, their decision spaces are usually smaller than those of GCSes. Since the information from a single point is usually limited (Pensieve only has 25 states [53]), the internal decision logic are usually not very complex. For example, Pensieve [53] and AuTO [18] use less than ten hidden layers to outperform carefully designed heuristics [18, 53]. The samll decision space enables us to explain the behaviors of policies in LCSes as a whole. Thus, we explain LCSes by converting DNNs into simpler models. In this paper, TranSys chooses decision tree as the target model due to its strong expressivity and similarity towards network policies [§4.1]. Moreover, the actions in networked systems usually have dependency over each other, while decision tree training requires inputs to be independent and identically distributed (i.i.d.). Thus, directly training decision tree from DNNs is impractical. We introduce an imitation learning-based resampling and training method in §4.2.

For GCSes, their decision-making complexity comes from two aspects. First, since GCSes have access to global information, the amount of information to process is much larger (e.g., tens or hundreds of nodes in Spark scheduling [18], hundreds of switches in a topology [64]). Second, GCSes usually need to take the graph structure as input, as presented in Figure 4, which is known difficult to be efficiently processed. Thus, the decision logic of GCSes is more challenging to explain when they are further coupled with DNNs. Our intuition here is that we could explain GCSes by finding their common features and unifying them into the same model. Thus, we propose hypergraph-based modeling. We find that most GCSes could be modeled with hypergraphs due to the graph-based essence of networked systems, which enables us to explain their internal behaviors specifically with hypergraph-based techniques. We introduce the modeling and explanation methods for GCSes in §5.

With the help of the above methods, we can address the drawbacks in §2.2. If the decision-making process of DL-based networked systems is interpretable, network operators can understand how each decision is made. When a system fails to achieve the expected performance, operators could troubleshoot the system by analyzing the explanations on why it makes such a decision. Besides, network operators could perform daily operations with the help of suggestions from explanations. Furthermore, TranSys converts DNNs of LCSes into *lightweight* decision trees, which could reduce the resource consumption and decision latency of LCSes.

4 Local Control Systems

We first analyze the reasons why we select decision trees as the conversion target [§4.1], and then introduce the methodology to convert the DNNs in LCSes to decision trees [§4.2].

4.1 Design Choice: Decision Tree

As introduced in §1, TranSys converts DNNs into transparent models based on interpretation methods. There are many candidate models such as (super)linear regression [34, 61], decision trees [12], and FSMs [44], etc. We refer the readers to [5] for a comprehensive understanding.

In this paper, among all candidate models, we decide to convert DNNs to *decision trees* due to three reasons. First, the logic structure of decision trees resembles the policies made by networked systems, which are usually logical combinations of several judgments. For example, bitrate selection of ABR algorithms depends on a comprehensive analysis over current buffer occupancy, last bitrate selection, and predicted throughput [53, 69, 77]. Second, decision trees have rich expressiveness and high faithfulness because they are non-parametric and can represent very complex policies [15]. We demonstrate the performance of decision trees during conversion compared to other methods [34, 61] in Appendix C.1. Third, decision trees are lightweight for networked systems, which will bring further benefits on resource consumption and decision latency [§7.2]. There are also promising research efforts that interpret deep RL with symbolic programming [49] or program search [73]. However, both of them need *expert knowledge* to be integrated and are domain-specific thus not taken in TranSys.

With explanations of LCSes in the form of decision trees, we can explain the results since the decision-making process is transparent now [§6.2]. Furthermore, we could also troubleshoot problems of DNN models when DNN models generate sub-optimal decisions [§7.1]. Moreover, since decision trees are much smaller in size, less expensive on computation, we could also deploy the decision trees online instead of DNN models. This will result in low decision-making latency and resource consumption [§7.2].

4.2 Conversion Methodology

Reinforcement Learning (RL) Assumption. Most DL-based LCSes adopt RL as their decision model [18, 40, 53]. The reason behind the wide adoption of RL in networked systems is the essence of the sequential decision-making process of networked systems. RL caters the need of LCSes and optimizes the overall policy instead of a single action. We provide a primer of RL in Appendix A.2. To convert the DNN models of LCSes more faithfully, we build TranSys atop RL. We also discuss the scenario when LCS is not RL-based at the end of this section.

Specifically, we adopt an imitation-learning-based method designed for the verification of deep RL [12], which also extracts decision trees from DNNs in RL. We apply it in networked systems and reproduce key steps of the conversion for network operators as follows:

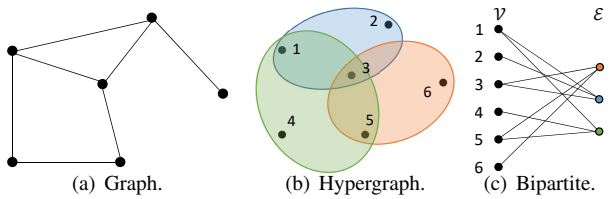
Step 1: Training and traces collecting. When network operators want to deploy a DL-based LCS into their environment, they first need to train that LCS online or in simulation environments [18, 53]. TranSys follows this procedure and collects the (state, action, reward) tuples into dataset $\mathscr{D}$ when the training process finishes.

Step 2: Resampling. Decision tree algorithms are designed in the scope of supervised learning, which usually requires the inputs of algorithms to be i.i.d. However, the tuples in $\mathscr{D}$ have dependencies on each other. Moreover, as RL optimizes the future accumulated reward of a *policy* while supervised learning optimizes the accuracy of a *single action*, their loss functions do not match. We need to resample $\mathscr{D}$ to make it suitable for decision tree training. We resample each tuple (s,a) with the ratio of two loss functions:

$$p(s,a) \propto \left(V^{(\pi^*)}(s) - \min_{a' \in A} Q^{(\pi^*)}(s,a')\right) \cdot \text{sign}\left((s,a) \in \mathscr{D}\right) \quad (1)$$

where $p(s,a)$ is the resampling probability for tuple (s,a), which might be equally scaled due to normalization. $V(s)$ and $Q(s,a)$ are the value function and Q-function of RL [71]. Value function represents the expected total reward starting at state s and following the policy π while Q-function further specifies the next step action a, which are introduced in Appendix A.2 in detail. π^* is the DNN policy and A is the action space. $\text{sign}(x)$ is the indicator function, which equals to 1 if and only if x is true.

Step 3: Conversion. We then train decision trees in two stages. In the first stage, we follow [12] and train the student with the resampled dataset. Our subtlety is in the second stage. As the size of the decision tree is sometimes much larger than network operators can understand, we adopt cost complexity pruning (CCP) [29] to reduce the number of leaf nodes according to the requirements from network operators. Compared with other pruning methods, CCP usually achieves a much smaller decision tree with similar error rate [56]. Smaller decision trees are easier to understand for network operators. In detail, CCP penalizes the cost with the complexity of decision tree and prunes subtrees to balance between the accuracy and complexity. Moreover, for the continuous outputs in networked systems (e.g., queue thresholds [18]), we employ the design of *regression tree* to generate real value outputs [72].



(a) Graph. (b) Hypergraph. (c) Bipartite.

Figure 5: In a hypergraph, one hyperedge can connect multiple vertices. (c) is the bipartite of hypergraph (b), where the left part denotes vertices and right part denotes hyperedges.

Step 4: Deployment. Finally, network operators could deploy the converted model online and enjoy both the performance improvement brought by deep RL and the transparency provided by the converted model.

From our experiments, decision trees with hundreds of leaf nodes can faithfully represent the original DNN-based policies [§6.2]. Moreover, if the DL-based LCS does not adopt RL but supervised/unsupervised learning instead, TranSys will skip ***Step 2*** and directly generate the decision tree with the collected traces $\mathscr{D}$. This is because models in both cases are trained with single actions, which means that tuples do not have dependency on each other. Furthermore, besides the decision tree algorithm we adopted as above, there are also promising research efforts that verify deep RL with, e.g., programming languages [73, 82]. However, they usually need *specific expert knowledge* and *human efforts* to pre-define some primitives and are difficult to be generalized. We leave them as our future work.

5 Global Control Systems

We first briefly introduce hypergraph and hypergraph neural network (HGNN) [§5.1], then present several applications on how to model networked systems with hypergraphs [§5.2], and finally introduce our explanation methods to find significant elements in hypergraphs [§5.3].

5.1 Hypergraph Primer

As introduced in §3, hypergraph is the mathematical foundation of global control systems in TranSys. Now we present the hypergraph-based model in detail.

Hypergraph. A hypergraph G is specified as follows:

$$G = < \mathscr{V}, \mathscr{E}, I > \quad (2)$$

where $\mathscr{V}$ and $\mathscr{E}$ denote the set of vertices and hyperedges. Incidence matrix $I^{|\mathscr{V}|\times|\mathscr{E}|}$ is a 0-1 matrix to represent the

Table 2: Several hypergraph-based models in different scenarios.

Scenario	Vertex	Hyperedge	Meaning of $H_{ve}=1$	Details
Global Traffic Optimization	Physical link	Path (src-dst pairs)	Path e contains link v.	§ 5.2.1
Cluster Job Scheduling	Job node	Dependency	Dependency e is related to node v.	Appendix B.1
Ultra-dense Cellular Network	Mobile user	Base station coverage	Base station e covers user v.	Appendix B.2
NF Consolidation & Placement	Physical server	Network function (NF)	One instance of NF e is on server v.	Appendix B.3

connection relationship between vertices and hyperedges. $I_{ve}=1$ indicates hyperedge e contains vertex v. We denote $V(e_i)$ as the set of all vertices connected to hyperedge e_i and $E(v_i)$ as the set of all edges connected to vertices v_i. We also have F_V and F_E as the features of vertices and hyperedges.

As presented in Figure 5(a), in a simple graph, one edge can only connect two vertices. In a hypergraph, hyperedges can connect more than two vertices, as shown in Figure 5(b). In this way, hypergraphs can represent more complicated scenarios, which is quite essential for the high-order information in GCSes (later presented in Table 2). To clearly illustrate the connections between vertices and hyperedges, hypergraphs could also be expressed as bipartites[3], as shown in Figure 5(c).

Hypergraph Neural Network (HGNN). Base on the introduction of hypergraph, we extend GNN to HGNN. The difference between HGNN and GNN is that HGNN has two-way message passing. In a GNN, messages are passed between vertices: Each vertex has the information of all its neighbor vertices. In contrast, in an HGNN, messages are passed between hyperedges and vertices. Each vertex has the information of all connected hyperedges and vice versa:

$$M_{v_i} = f\left(\left\{M_{e_j} | e_j \in E(v_i)\right\}\right) \tag{3}$$

$$M_{e_i} = g\left(\left\{M_{v_j} | v_j \in V(e_i)\right\}\right) \tag{4}$$

where M_{v_i} and M_{e_i} are the embedded messages of vertex v_i and hyperedge e_i respectively, which are usually initialized as F_{v_i} and F_{e_i} [Appendix A.1]. f and g are transition functions, which are usually represented with DNNs.

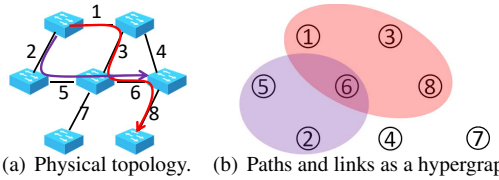
5.2 Model Applications

We present the summary of hypergraph representations of several typical scenarios in Table 2. Among them, we introduce the details of an SDN-based global traffic optimizer in Section 5.2.1 and other applications in Appendix B.

5.2.1 Global Traffic Optimization Model

SDN controller can collect the information from all data plane switches. In this case, a global traffic optimizer analyzes the traffic demands for each src-dst pair and generates the *routing paths* for all src-dst traffic demands based on the topology structure and link capacity [7, 64].

With the physical topology as a simple graph, compared to vertices (switches) and edges (links), paths are high-order information and are difficult to be efficiently expressed. Previous research efforts try to represent the paths with integer programming [23], which is hard to be efficiently optimized. RouteNet [64] designs a special GNN to optimize the routes, which is a special case of our HGNN-based model. We consider the paths as hyperedges, physical links as vertices, and transfer the problem into a hypergraph model. We embed the queuing delay on switches to a part of link latency in the hypergraph model.



(a) Physical topology. (b) Paths and links as a hypergraph.

Figure 6: The hypergraph representation of the global traffic optimization model.

An illustration of hypergraph mapping results is shown in Figure 6. Links (1,2,$\cdots$,8) are modeled as vertices, and paths (2→5→6, 1→3→6→8) are modeled as hyperedges in the hypergraph. Vertex features F_V could be the link capacity. Hyperedge features F_E could be the traffic demands for each transmission pair. The traffic optimizer will then continuously select the best routing paths for each src-dst traffic demand pair according to the varying traffic demand.

5.3 Hypergraph Explanations

Since different vertex-hyperedge connections contribute differently to model outputs, our design goal is to find a set of significant connections that have maximal influences on the results. Thus, we maximize the similarity between the results Y_W generated with those significant connections and results Y from the original hypergraph. To efficiently search for significant connections, we allow a fractional incidence matrix $W \in [0,1]^{|\mathcal{V}|\times|\mathcal{E}|}$ as a mask, which satisfies:

$$\mathbf{0} \leqslant W_{ve} \leqslant I_{ve},\ \forall v \in \mathcal{V}, e \in \mathcal{E} \tag{5}$$

Mask matrix W represents the significance of each connection contributing to the output in the hypergraph. A higher mask value W_{ve} indicates that the connection between vertex v and hyperedge e is more significant to the output.

A good mask W should have the following properties:

Faithful. The results generated by the mask should be as faithful as possible to the original results. The faithfulness requirement is to ensure that the connections found by W are the critical components to the output. To measure the

[3] A bipartite is a graph whose vertices are two disjoint sets [2].

similarity between outputs generated from I and W, we adopt Kullback-Leibler divergence [46] as the metric for discrete outputs and mean square error for continuous outputs.

Small. The number of significant connections should also be small enough to be understandable for network operators. If the algorithm provides too many "significant" connections, network operators will be confused and cannot easily explain the networked systems. Thus, we also need to optimize the scale of mask W.

Confident. Moreover, we also expect the results of W to be *confident*. For each connection (v,e), it is either seriously suppressed (W_{ve} close to 0) or almost unaffected (W_{ve} close to 1). In this paper, TranSys optimizes the entropy of mask W to encourage the connections in W to be close to 1 or 0.

However, some of the design goals above may conflict with each other. For example, less significant connections may leave out some important information of the original hypergraph, which leads to a different output from the original one. Thus we design a loss function to balance the three optimization goals above:

$$\ell(W) = D(W,I) + \lambda_1 ||W|| + \lambda_2 H(W) \tag{6}$$

where

$$D(W,I) = \sum Y_W \log \frac{Y_W}{Y} \tag{7}$$

$$||W|| = \sum_{v,e} |W_{ve}| \tag{8}$$

$$H(W) = -\sum_{v,e} (W_{ve} \log W_{ve} + (1-W_{ve}) \log(1-W_{ve})) \tag{9}$$

Among them, $D(W,I)$ is the KL-divergence, $||W||$ is the scale of the matrix, and $H(W)$ is the entropy of W. λ_1 and λ_2 are two hyperparameters to balance the optimization goals above. Network operators can adjust the weights of different parts of the optimization objective according to their needs. Thus the optimization problem can be formally written as:

$$\arg\min_W \ell(W) \;\text{ s.t. }\; Eq.(5);\ \lambda_1 > 0;\ \lambda_2 > 0 \tag{10}$$

To efficiently train and obtain the optimized mask W, we need to adopt training methods such as stochastic gradient descent (SGD). As a basic algorithm in the machine learning community, SGD has achieved numerous successful applications in different models [83]. However, since W is bounded in $[0,1]$ but not the whole real space $\mathbb{R}^{|V|\times|E|}$, SGD is not directly applicable. In response, we introduce a small trick to generate W. We construct a real matrix $W' \in \mathbb{R}^{|V|\times|E|}$ and get mask W by the following equation:

$$W = I \circ \text{sigmoid}(W') \tag{11}$$

$\circ$ means element-wise multiplication and sigmoid function is applied to each element separately. As there is no constraint on W', we can use SGD algorithm to optimize W directly.

In this way, we can obtain the mask value for each connection in a hypergraph. The mask value represents how significant the connection contributes to the output. We present the meaning and use cases of the mask values in §6.3 and §7.3 based on the SDN traffic optimization scenario.

6 Implementation and Evaluation

We first introduce implementation details of TranSys and three DL-based networked systems [§6.1]. We then present the explanations of two LCSes [§6.2] and one GCS [§6.3] with TranSys. We leave the use cases of TranSys to §7.

6.1 Implementation Details

We implement two LCSes, Pensieve [53] and AuTO [18], and one GCS, RouteNet* [64], in our experiments. We apply TranSys over them respectively.

6.1.1 Pensieve Implementation

In current Internet video transmissions, each video consists of multiple *chunks* (a few seconds of playtime) and each chunk is encoded at multiple bitrates [70]. Client-side video players employ ABR algorithms to optimize the QoE of users against the bandwidth heterogeneity and spatio-temporal variations [53, 76]. Pensieve is a deep RL-based ABR system to optimize bitrates with network observations such as past chunk throughput, buffer occupancy.

We use the same video in Pensieve unless other specified: a sample video with a total length of 193 seconds. The chunk size, bitrates of the video are respectively set to 4 seconds and $\{300, 750, 1200, 1850, 2850, 4300\}$ kbps. Real-world network traces include 250 HSDPA traces [63] and 205 FCC traces [22]. We set up the same environment with Pensieve. We integrate DNNs into JavaScript with `tf.js` [67] to run Pensieve in the browser.

We use *linear QoE* as our optimization objective:

$$QoE = \frac{1}{N}\left(\sum_{n=1}^{N} R_n - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} |R_{n+1} - R_n|\right) \tag{12}$$

At the n^{th} chunk among N chunks, R_n and T_n represent the bitrate and rebuffer time. Three terms in Equation 12 refer to video quality, rebuffer penalty, and smoothness penalty. Results for other types of QoE are similar. In all our experiments, we use the finetuned model provided by [53].

We then implement TranSys-over-Pensieve (ToP). We implement the decision tree training with `scikit-learn` [60] and modify it to support the Cost Complexity Pruning. We train decision trees on one server equipped with an Intel Core i7-8700 CPU (6 physical cores) and an Nvidia Titan Xp GPU. We set the number of samples after resampling to 200k and randomly divide them to the training set (80%) and

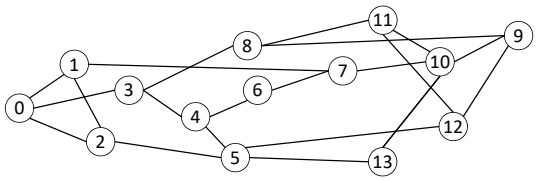


Figure 7: NSFNet topology [4].

test set (20%). We set the number of leaf nodes to 200 based on several experiences. We further discuss the setting of leaf nodes in Appendix C.2.

We also use five baselines (BB [38], RB [53], FESTIVE [41], BOLA [69], rMPC [77]) developed in recent years and migrate them into `dash.js` client [1] based on the implementations of baselines provided by [53].

6.1.2 AuTO Implementation

AuTO is a scalable flow scheduling system to optimize FCT based on deep RL. Limited by the long decision latency of DNN, AuTO can only optimize long flows individually with long-flow RL agent (lRLA). For short flows, AuTO makes decisions locally with multi-level feedback queue [11] and optimizes the queue thresholds with short-flow RL agent (sRLA). lRLA takes {5-tuples, priorities} of active long flows, and {5-tuples, FCTs, flow sizes} of finished long flows as states and decides the {priority, rate limit, routing path} for each active long flow. sRLA observes {5-tuples, FCTs, flow sizes} of finished short flows and outputs the queue thresholds.

We use the same 16-server one-switch topology and traces evaluated in AuTO: a web search (WB) [33] trace and a data mining (DM) [8] trace. We train the DNNs for 8 hours following the instructions in [18]. We use two H3C-S6300 48-port switches. All other configurations (e.g., link capacity, link load, DNN structure) are set the same with AuTO. We then evaluate TranSys-over-AuTO (ToA). For two DNNs in AuTO (lRLA and sRLA), we set the number of leaf nodes to 2000. This is because the state spaces of lRLA (143 states) and sRLA (700 states) are much larger than that of ToP (25 states). Moreover, the robustness of this parameter will also be demonstrated in Appendix C.2.

6.1.3 RouteNet* Implementation

RouteNet is a centralized traffic optimization system in the SDN scenario [64]. RouteNet collects the underlying physical topology and the traffic demand matrix as input and predicts end-to-end latency on each path (src-dst pair) to provide information for routing decisions. To deal with the interaction between links and paths, RouteNet adopts GNN to generate predictions and redesigns the training methods for routing paths, which could be formulated with hypergraphs in TranSys as introduced in §5.2.1.

We adopt the same GNN architecture as RouteNet [64]

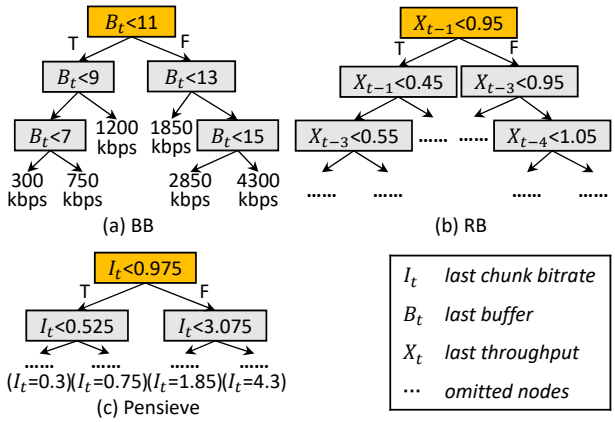


Figure 8: Decision tree representations of ABR algorithms.

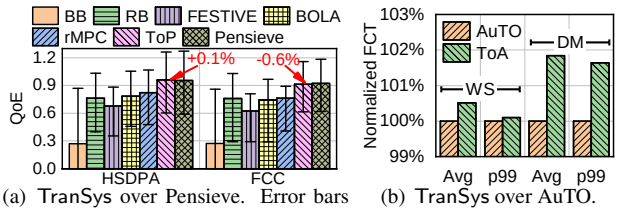
and refactor it with the hypergraph model of TranSys. We retrain the model on a dataset composed of different traffic patterns, routing schemes, and corresponding end-to-end delay. The dataset is generated by OMNET++ simulations on a well-known backbone topology NSFNet [4] and provided by RouteNet. The topology is presented in Figure 7. We adopt the close-loop routing system RuoteNet*, which concatenates path latency predictions with routing decisions.

We then apply the hypergraph-based explanation method of TranSys over RouteNet*. We set $\lambda_1 = \lambda_2 = 0.25$ according to several empirical experiments. Network operators can adjust the hyperparameters according to their preferences. We also discuss the sensitivity of these two hyperparameters in Appendix D. We optimize the mask matrix with SGD as introduced in §5.3.

6.2 Explaining LCS

We evaluate the performance of TranSys over LCSes by qualitatively explaining the new policies learned by DNNs and quantitatively measuring the performance loss between the decision tree and the original DNN. Both evaluation results demonstrate that TranSys could verify human's knowledge, unveil some undiscovered insights, and faithfully convert the DNNs with $<2\%$ performance loss.

Policy analysis. We first explain the behaviors of Pensieve. We present a part of the decision trees extracted from Pensieve, BB, and RB in Figure 8. As shown in Figure 8(a) and 8(b), the results of current heuristic methods are dominant either by the last buffer occupancy or by the throughput from last several chunks. This is quite different from the policies learned by Pensieve. As shown in Figure 8(c), at the root node, Pensieve first classifies the states based on their last bitrate. Further policies are then generated for each branch (omitted). Thus, by converting DNNs in Pensieve into decision trees, we observe a new decision variable in ABR algorithm design: the information contained in the last bitrate may be more than we thought. This observation provides a different way to design heuristic ABR algorithms:



(a) TranSys over Pensieve. Error bars represent 25%ile and 75%ile.

(b) TranSys over AuTO.

Figure 9: Faithfulness of TranSys.

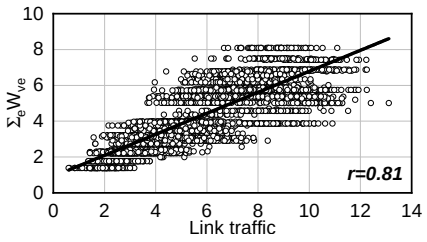


Figure 10: The correlation between $\sum_e W_{v,e}$ and link traffic.

Table 3: Top 5 mask value explanations in Figure 7.

	Routing path	Link	Mask value	Explanation type
1	6→7→10→9	6→7	0.791	Shorter
2	1→7→10→9	1→7	0.772	Shorter
3	7→10→9→12	10→9	0.751	Less congested
4	8→3→0→2	8→3	0.746	Shorter
5	6→4→3→0	6→4	0.746	Less congested

network operators could design different ABR algorithms customized for each last chunk bitrate.

Performance faithfulness. We further show that the application-level performance of TranSys-based systems is faithful to the original systems for both Pensieve and AuTO. As shown in Figure 9(a), the differences in average QoE between the decision tree generated by TranSys (ToP) and Pensieve on two traces are less than 1%. Furthermore, we also measure the performance ratio of the decision tree generated from AuTO (ToA). As shown in Figure 9(b), ToA has a performance loss within 2% compared to AuTO. The performance loss of both systems is much less than the gain of introducing machine learning (Pensieve by 14%, AuTO by up to 48%). TranSys have comparable performance to DNNs with decision trees only. We also implement two regression-based target models (linear regression [61] and mixed regression [34]) and compare their performance against TranSys in Appendix C.1. Decision tree outperforms the other two target models, which confirms our design choice in §4.1.

Sensitivity and overhead. We demonstrate in Appendix C.2 that TranSys is robust to a wide range of hyperparameters (number of leaf nodes in this case): TranSys performs well in a wide range from 200 to 5000 leaf nodes. Thus, network operators do not need to carefully finetune TranSys. Our evaluation results also show that translating finetuned DNNs into decision trees needs negligible offline computation time of less than one minute, which is much less than the training time of DNN models [Appendix C.3].

6.3 Explaining GCS

Explaining how the DL-based routing system makes decisions will make the system more trustworthy for network operators. We first present several examples to explain the behaviors of RouteNet* by analyzing the mask values generated by TranSys and then show the overall behaviors of the explanations provided by TranSys.

Individual explanations. We apply TranSys over the topology in Figure 7 and follow the implementations in §6.1.3. The top-5 mask values and their paths and links are presented in Table 3. Usually, there are two possible reasons behind a routing agent selecting path a instead of path b:

(i) Path a is shorter than path b. For example, for the path 1 in Table 3, the connection of path 6→7→10→9 and link 6→7 has a high mask value of 0.791, indicating selecting 6→7 is a significant decision for that path. This is because there are three candidate paths that are less than 4 hops: p_1^1: 6→7→10→9; p_2^1: 6→4→3→8→9; and p_3^1: 6→4→5→12→9, as shown in Figure 7. The shortest path p_1 has the first hop of 6→4 while the other two have 6→7. Thus TranSys discovers that selecting the first hop is important in deciding the route from 6 to 9 and 6→7 is selected for a shorter path. For other paths in the subsequent decisions, since they are much longer than p_1^1, they will be abandoned by RouteNet* in the early stage. Thus, once the first hop is decided, the following path is naturally decided.

(ii) Path a is less congested than path b. For example, for the path 3 in Table 3, there are two paths with the same length: p_1^3: 7→10→9→12; and p_2^3: 7→10→11→12. TranSys also correctly identifies the significant branch and finds that 10→9 is an important decision. Note that with the help of HGNN, the information of the following path (hyperedge) is embedded into each connected link (vertex) according to Equation 3 and 4. Thus the DL-based routing agent selects the p_2^2 for congestion avoidance and the significant branch (10→9 or 10→11) is identified by TranSys.

Overall behaviors. Besides the individual explanations over connections, we also analyze the overall behaviors of TranSys to demonstrate the interpretability and causality of TranSys. We sum up all mask values on each link (the vertex in the hypergraph) $\sum_e W_{v,e}$, and measure their relationship with the traffic on each link. As shown in Figure 10, the sum of mask values and link traffic have a Pearson's correlation coefficient of $r = 0.81$. Thus, the sum of mask values and link traffic are statistically significantly correlated, which indicates that the explanations provided by TranSys are reasonable. Note that TranSys can provide connection-level explanations as presented above, which is finer-grained than the information from link traffic.

Sensitivity and overhead. We present how different optimization goals in §5.3 react to the changes of hyperparameters λ_1 and λ_2 in Appendix D. Our results demonstrate that network operators could efficiently emphasize different parts

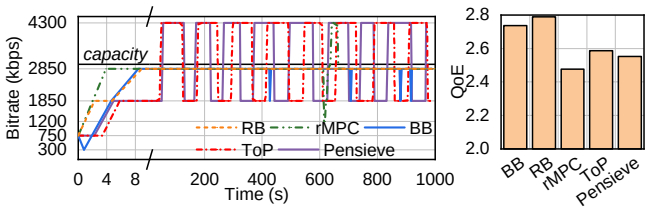


Figure 11: Bitrate decisions on a 3000kbps link. BB, RB and rMPC converge to 2850kbps while Pensieve and ToP oscillate between 1850kbps and 4300kbps.

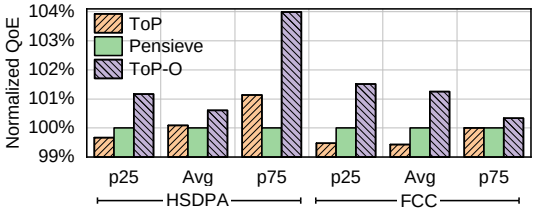


Figure 12: QoE of ToP with oversampling (ToP-O) normalized by Pensieve.

of optimization goals in §5.3 by adjusting respective hyperparameters. Besides, the computation overhead is also quite acceptable: for all hypergraph explanation experiments, our computation time of mask matrices is less than 100 seconds.

7 Use Cases

In this section, we demonstrate three use cases of TranSys in different aspects of networked systems:

- **Troubleshooting networked systems.** With transparent explanations, TranSys successfully troubleshoots a problem of Pensieve and improves its performance by adjusting the structure of decision trees [§7.1].
- **Lightweight deployment.** With the high faithfulness of explanations provided by TranSys, network operators could directly deploy the converted decision trees online and enjoy the lightweight benefits brought by decision trees [§7.2].
- **Rerouting optimization.** We provide a preliminary case study on how to estimate and select the optimized path based on the mask values provided by TranSys when network operators want to reroute a certain path [§7.3].

7.1 Troubleshooting Pensieve

We show a use case to troubleshoot the sub-optimal decisions in Pensieve with TranSys. When converting ToP, we observe that some bitrates are rarely selected by Pensieve: For experiments in §6.2, among six bitrates from 300kbps to 4300kbps, two bitrates (1200kbps and 2850kbps) are rarely selected by Pensieve (<0.1%). Details are in Appendix E.1. The imbalance selection raises our interests since missing bitrates are *intermediate* bitrates: the highest or lowest bitrates may not be selected due to network conditions, but not intermediate ones.

We further emulate Pensieve on a link with bandwidth fixed to 3000kbps, the optimal decision of which is constantly selecting 2850kbps[4]. Bitrate selections on the 3000kbps link at different time are presented in Figure 11. Decisions of Pensieve oscillates between 1850kbps and 4300kbps, which is also mimicked by ToP faithfully but are sub-optimal. In contrast, other baseline algorithms learn the optimal selection policy and fix their decisions to 2850kbps, achieving a higher QoE. Similar observations can also be observed when the link bandwidth are fixed around 1200kbps. Further investigations show that Pensieve does not have enough confidence in either choice [Appendix E.2].

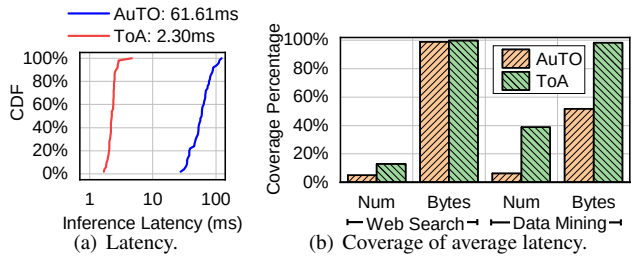
The training mechanism of Pensieve possibly causes this problem. At each step, the agent tries to *reinforce* particular action that leads to a larger reward. In this case, when the agent discovers that four out of six actions can achieve a relatively good reward, it will keep reinforcing this discovery by continuous selecting those actions and finally abandon the other two actions. Making decisions with fewer actions brings higher confidence to the agent, but also makes the agent converge to a local optimum in this case.

Without TranSys, as the decision-making process inside DNNs is nontransparent to network operators, the only thing they can do is to spend several hours to days in retraining the model and encourage the exploration of deep RL. However, converting it into decision trees enable us to fix this problem by adjusting its internal structure manually. Since the dataset $\mathscr{D}$ to train the decision tree is highly *imbalanced*, as a straightforward solution, we oversample the missing bitrates to make sure their frequencies after resampling are around 1%. We normalize the QoE over the QoE of Pensieve and present the results in Figure 12. The oversampled ToP (ToP-O) again outperforms DNNs with decision trees by around 1% on average and 4% at the 75^{th} percentile on HSDPA traces. Further improvement with elaborate designs (e.g., using imbalanced decision trees [6] to make decisions more balanced) are possible since the decision-making process is transparent to us, which is left for future work.

7.2 Lightweight Deployment

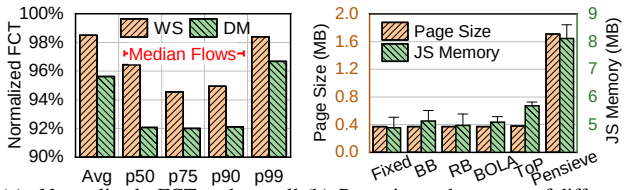
The second use case we want to demonstrate is how to use TranSys to make LCSes lightweight in practical deployment. Decision trees provided by TranSys are not only easy to explain but also lightweight to deploy. We demonstrate that directly deploying decision trees online in practical scenarios will (*i*) shorten the decision latency and also (*ii*) reduce resource consumption, which will bring further benefits.

[4]The bandwidth is a little higher than the bitrate of 2850kbps due to the difference between goodput and link bandwidth.



(a) Latency. (b) Coverage of average latency.

Figure 13: The latency between flow servers send states to and receive actions from the RL server.



(a) Normalized FCT when all flows are reported upward. (b) Page size and memory of different ABR algorithms.

Figure 14: The lightweightness benefits brought by TranSys.

7.2.1 Decision Latency

Since the decision period of Pensieve is usually 2-4s and not a critical issue, we discuss the decision latency of AuTO here. AuTO designs different processing strategies for short flows and long flows. This is because its per-flow decision-making latency (lRLA) is close to 100ms, during which short flows in datacenters will run out. Converting the DNNs into decision trees enables us to make precise per-flow decisions for more flows since the decision latency is shortened. As shown in Figure 13(a), when replacing DNNs with decision trees, the decision latency could be reduced by 26.8×. In this case, ToA will cover 39% of flows and 98% of bytes while the coverage of AuTO is 6% and 52% for DM traces [18], as shown in Figure 13(b). This further results in two benefits:

Performance Benefits. Now we can perform per-flow scheduling with decision trees for not only long flows but also *median flows*. As a result, increasing the coverage of precise per-flow control will improve the overall performance. We implement a prototype of a high-coverage per-flow control algorithm based on ToA. By replacing the DNN with the decision tree generated from TranSys, more median flows can receive the scheduling actions and thus follow the optimized per-flow scheduling strategies. We present the FCT results normalized by those of AuTO in Figure 14(a). Although the decision tree has not experienced the scheduling of median flows during training, it can still improve the average performance by 1.5% and 4.4% for two traces. We can also observe significant performance improvement for those median flows (from the 50^{th} to the 90^{th} percentile) by up to 8.0%. This indicates that median flows enjoy the per-flow precise scheduling benefits. The performance improvement of DM is more significant than WS since the coverage increased by applying decision trees of DM is larger than that of WS (cf. Figure 13(b)).

Implementation Benefits. Another benefit of converting DNNs into decision trees is that they can be implemented on devices with limited programmability. The DNNs in AuTO are very difficult to be implemented with low-level languages such as P4 [16] since there are a lot of complicated operations (e.g., float number, sigmoid activation). In contrast, decision trees could be implemented with P4 by converting the branching logic into match-action tables. This enables the deployment of decision trees onto *SmartNICs* [59] and *programmable switches* [16], which could offload the whole decision-making process to data plane and increase the coverage of per-flow scheduling to *all flows*. We preliminarily demonstrate the practicality by implementing the decision tree onto a Netronome NFP-4000 SmartNIC [59] with P4[5]. We deploy the ToA-lRLA decision tree and measure the decision-making latency. Evaluation results show that the decision-making latency is only 9.37μs on average. The latency might be further reduced with programmable switches. We leave the deployment of decision trees on programmable switches [16] and the comparison with other baselines for future work.

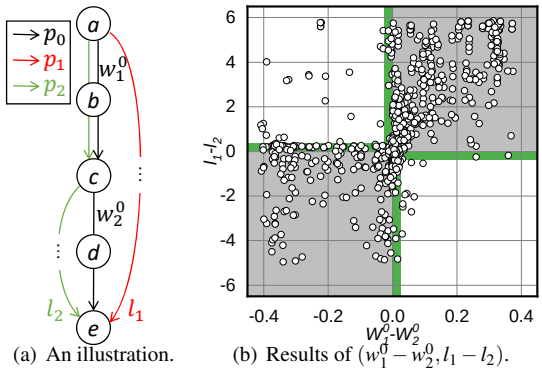
7.2.2 Resource Consumption

We then evaluate the resource consumption benefits of ToP. As there are other functions in the player (e.g., initialization), we compare ABR algorithms with a *fixed bitrate selection* algorithm, which always selects the lowest bitrate. We evaluate the ABR algorithms on their initial page load time and runtime memory consumption:

Page load time. If the HTML page size is too large, users have to wait for a long time before the video starts to play. As shown in Figure 14(b), Fixed, BB, RB, and BOLA have almost the same page size since their processing logic are simple. Pensieve increases the page size by 1370KB since it needs to download the DNN model first. In contrast, ToP has a similar page size with the heuristics. When the goodput is 1200kbps (the average bandwidth of Pensieve's evaluation traces), the *additional* page load time of ABR algorithms is reduced by 156×: Pensieve introduces an additional page load time of 9.36 seconds $\left(\frac{(1750-380)KB}{1200kbps}\right)$, while ToP only adds 60ms $\left(\frac{(389-380)KB}{1200kbps}\right)$.

Runtime memory consumption. We then measure the runtime memory and present the results in Figure 14(b). Due to the complexity of forward propagation in the neural networks, Pensieve consumes much more memory than other ABR algorithms. In contrast, the additional memory introduced by ToP is reduced by 4.0× on average and 6.6× on the peak, which is at the same level as other heuristics.

[5]Compared with programmable switches (e.g., Barefoot Tofino), SmartNICs have more adequate resources and less hardware limitations. More implementation details could be found at `https://github.com/TranSys2020/TranSys/tree/master/case_2`.



(a) An illustration. (b) Results of $(w_1^0 - w_2^0, l_1 - l_2)$.

Figure 15: Selecting the best path to reroute.

7.3 Optimized Rerouting Estimation

We present a potential use case of TranSys on selecting candidate paths during rerouting. As shown in Figure 15(a), when the original path p_0 from node a to node e needs rerouting, there are several candidates that might be suitable for the traffic demand from a to e (p_1 and p_2). Since network operators do not know the performance until rerouting is finished, deciding which path to reroute is challenging. This scenario is especially common in topologies such as *fat trees* where there are multiple equal-cost paths from one node to another.

Our observation is that since the candidate paths divert at different nodes from the original path, we could estimate their performance by the mask value of the connection between the divert node and its next-hop link. For example, in Figure 15(a), p_1 diverts from p_0 at node a while p_2 diverts from p_0 at node c. w_1^0 is the mask value of the connection between p_0 and link $a \to b$. Since w_1^0 represents the significance of selecting $a \to b$ rather than other links, it is correlated to the possibility that there is also a relatively good path if $a \to b$ is not selected. A lower mask value of a connection means that the selection is not important in deciding the routing path. Thus we have:

OBSERVATION. *If $w_1^0 > w_2^0$, the latency of p_1 (denoted as l_1) is likely to larger than the latency of p_2 (denoted as l_2).*

We verify the observation above with the NSFNet topology in Figure 7. Since the optimal path may not be the shortest path, we consider a path as a candidate if it is $\leqslant$1 hop longer than the original path. For example, for path 0→2→5→12, 0→3→4→5→12 is considered as a candidate but 0→1→7→10→11→12 is not. We go through all pairs of demand in the NSFNet topology in Figure 7 and measure all the path latency for such candidate scenarios and the mask values at the diverting node. We repeat the experiments at all the 50 traffic samples provided by [64].

As a case study, for each routing path p_0 generated by RouteNet*, we connect all tuples (p_0, p_1, p_2) that satisfy the conditions in Figure 15(a), and measure their end-to-end latency (l_0, l_1, l_2). We also measure the mask values at the diverting nodes (w_1^0 and w_2^0) and plot the $(w_1^0 - w_2^0, l_1 - l_2)$. For simplicity, we present the results of all paths originating from nodes 0,1,2,3 in Figure 15(b) (750 points in total). Most points (72%) fall into quadrant I and III (shadowed gray) with another 19% points very close to quadrant I and III (shadowed green), which verifies our observation above. Thus, we provide an indicator for network operators to efficiently decide which path to reroute without estimating end-to-end path latency.

8 Discussion

We discuss the limitations and future directions of TranSys. We also discuss two open problems (fairness and dynamics) of deploying DL-based networked systems in Appendix F.

Explanation methods. TranSys employs two explanation methods in this paper: decision tree-based method for LCSes and hypergraph-based method for GCSes. However, we are not going to separate the DL-based networked systems into two disjoint sets strictly. Sometimes LCSes could also have graph-based information in their inputs. For example, in the OSPF routing, each node has the information of a spanning tree. When each node is enhanced with DNNs, we can also apply our graph-based explanation method. Thus, the explanation methods in TranSys should be freely selected by network operators according to the structure of inputs to their DL-based networked systems. We plan to investigate a detailed guideline to help network operators to appropriately select better explanation methods for their DL-based networked systems. A systematic approach may also need to be studied for automatic selection of explanation methods in TranSys for different DL-based networked systems.

Performance guarantees. Although TranSys achieves faithful explanations over many DL-based networked systems, TranSys can not theoretically guarantee the performance but only provide the explanations with best efforts. Since the internal structure of DNNs is too complicated to be understood by human experts, it is usually difficult for DL-based networked systems to provide a theoretical guarantee on their performance [3]. Adopting recent advances with the help of adversarial learning [37] would be a potential avenue to increasing the worst-case performance or even establishing a performance bound for DL-based networked systems, which is left as our future work.

9 Related Work

***(i)* Practicality of DL-based networked systems.** Some recent work focuses on the verification of DL-based networked systems [43] with advanced verification techniques [82], which is orthogonal to our work and could be adopted together for a more practical system. There are also some position papers that discuss the interpretability of DL-based networked systems [26, 81], which, however, remain preliminary on both problems and solutions. In contrast, TranSys

provides a systematic solution to effectively explain diverse DL-based networked systems with high quality for practical deployment, taking advantage of decision tree and hypergraph. There are also research efforts on building the training platform for DL-based networked systems [52]. TranSys could be integrated with those platforms to achieve a practical system at the design phase, which is left for future.

***(ii)* Explainability in other communities.** There are a number of research efforts that try to build explainable systems in different communities, including image analysis [13, 66, 80], natural language processing [19, 62], recommendation systems [17, 21], and security [30, 34]. However, most of them are designed for Convolution Neural Network (CNN) or Recurrent Neural Network (RNN) and are not suitable for HGNNs and decision processes in networked systems. An earlier work explains the discrete node classification results in GNNs with mutual information [78], which is orthogonal to our method. In contrast, the two-part explanations of TranSys could explain diverse DL-based networked systems, including both simple LCSes and HGNN-based GCSes.

10 Conclusion

In this paper, we propose TranSys, a new framework to explain DL-based networked systems. TranSys categorizes DL-based networked systems and provides respective solutions by modeling and analyzing the commonplaces of them. We apply TranSys over several typical DL-based networked systems. Evaluation results show that TranSys-based systems can explain the behaviors of DL-based networked systems with high quality. Further use cases demonstrate that network operators could adopt TranSys to efficiently troubleshoot DL-based networked systems, make DL-based networked systems more lightweight, provide suggestions during network operations for DL-based networked systems, and also bring many additional performance benefits. This work does not raise any ethical issues.

References

[1] Dash.js. https://github.com/Dash-Industry-Forum/dash.js, 2018.

[2] Bipartite graph - wikipedia. https://en.wikipedia.org/wiki/Bipartite_graph, 2019.

[3] Neurips 2019 workshop on machine learning with guarantees. https://sites.google.com/view/mlwithguarantees/, 2019.

[4] Nsfnet topology - knowledge-defined networking training datasets. http://knowledgedefinednetworking.org/, 2019.

[5] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence. *IEEE Access*, 6:52138–52160, 2018.

[6] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proc. AAAI*, 2019.

[7] Ian F Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30, 2014.

[8] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proc. ACM SIGCOMM*, 2010.

[9] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, et al. pfabric: Minimal near-optimal datacenter transport. In *Proc. ACM SIGCOMM*, 2013.

[10] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *Proc. USENIX NSDI*, 2018.

[11] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Information-agnostic flow scheduling for commodity data centers. In *Proc. USENIX NSDI*, 2015.

[12] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Proc. NeurIPS*, 2018.

[13] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proc. IEEE CVPR*, 2017.

[14] Albert Bifet, Jiajin Zhang, Wei Fan, Cheng He, Jianfeng Zhang, Jianfeng Qian, Geoff Holmes, and Bernhard Pfahringer. Extremely fast decision tree mining for evolving data streams. In *Proc. ACM KDD*, 2017.

[15] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1-2):285–297, 1998.

[16] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[17] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. Neural attentional rating regression with review-level explanations. In *Proc. WWW*, 2018.

[18] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proc. ACM SIGCOMM*, 2018.

[19] Minmin Chen. Minimalrnn: Toward more interpretable and trainable recurrent neural networks. In *Proc. NIPS Symposium on Interpretable Machine Learning*, 2017.

[20] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[21] Zhiyong Cheng, Xiaojun Chang, Lei Zhu, Rose C Kanjirathinkal, and Mohan Kankanhalli. Mmalfm: Explainable recommendation by leveraging reviews and images. *ACM Transactions on Information Systems (TOIS)*, 37(2):16, 2019.

[22] Federal Communications Commission. Raw data - measuring broadband america. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016, 2016.

[23] Rene L Cruz and Arvind V Santhanam. Optimal routing, link scheduling and power control in multihop wireless networks. In *Proc. IEEE INFOCOM*, 2003.

[24] Yann N Dauphin and Yoshua Bengio. Big neural networks waste capacity. In *Proc. ICLR*, 2013.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. IEEE CVPR*, 2009.

[26] Arnaud Dethise, Marco Canini, and Srikanth Kandula. Cracking open the black box: What observations can tell us about reinforcement learning agents. In *Proc. ACM NetAI*, 2019.

[27] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *Proc. ACM SIGCOMM*, 2011.

[28] Pierre Ecarlat. Cnn - do we need to go deeper? `https://medium.com/finc-engineering/cnn-do-we-need-to-go-deeper-afe1041e263e`, 2017.

[29] Jerome H Friedman, Richard A Olshen, Charles J Stone, et al. Classification and regression trees. *Belmont, CA: Wadsworth & Brooks*, 1984.

[30] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai^2: Safety and robustness certification of neural networks with abstract interpretation. In *Proc. IEEE S&P*, 2018.

[31] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: Enabling innovation in network function control. In *Proc. ACM SIGCOMM*, 2014.

[32] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proc. USENIX NSDI*, 2011.

[33] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *Proc. ACM SIGCOMM*, 2009.

[34] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. Lemna: Explaining deep learning based security applications. In *Proc. ACM CCS*, 2018.

[35] Moritz Hardt, Eric Price, Nati Srebro, et al. Equality of opportunity in supervised learning. In *Proc. NIPS*, 2016.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE CVPR*, 2016.

[37] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Proc. NIPS*, 2017.

[38] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. ACM SIGCOMM*, 2014.

[39] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proc. ACM MobiSys*, 2017.

[40] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *Proc. ICML*, 2019.

[41] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proc. ACM CoNEXT*, 2012.

[42] Mahmoud Kamel, Walaa Hamouda, and Amr Youssef. Ultra-dense networks: A survey. *IEEE Communications Surveys & Tutorials*, 18(4):2522–2545, 2016.

[43] Yafim Kazak, Clark Barrett, Guy Katz, and Michael Schapira. Verifying deep-rl-driven systems. In *Proc. ACM NetAI*, 2019.

[44] Anurag Koul, Sam Greydanus, and Alan Fern. Learning finite state representations of recurrent policy networks. In *Proc. ICLR*, 2019.

[45] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. Learning to optimize join queries with deep reinforcement learning. *arXiv:1808.03196*, 2018.

[46] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 03 1951.

[47] Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig. Deep learning with dynamic computation graphs. In *Proc. ICLR*, 2017.

[48] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proc. NIPS*, 2017.

[49] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proc. AAAI*, 2019.

[50] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297, 1967.

[51] Chaitanya Manapragada, Geoffrey I Webb, and Mahsa Salehi. Extremely fast decision tree. In *Proc. ACM KDD*, 2018.

[52] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Ravichandra Addanki, Mehrdad Khani, Songtao He, et al. Park: An open platform for learning augmented computer systems. In *Proc. NeurIPS*, 2019.

[53] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proc. ACM SIGCOMM*, 2017.

[54] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proc. ACM SIGCOMM*, 2019.

[55] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. In *Proc. VLDB*, 2019.

[56] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.

[57] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, et al. Playing atari with deep reinforcement learning. In *Proceedings of NIPS 2013 Workshop on Deep Learning*, 2013.

[58] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[59] Netronome. Wihte paper: Nfp-4000 theory of operation. `https://www.netronome.com/media/documents/WP_NFP4000_TOO.pdf`, 2016.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[61] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proc. ACM KDD*, 2016.

[62] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In *Proc. ACL*, volume 1, pages 856–865, 2018.

[63] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3g networks: Analysis and applications. In *Proc. ACM MMSys*, 2013.

[64] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Unveiling the potential of graph neural networks for network modeling and optimization in sdn. In *Proc. ACM SOSR*, 2019.

[65] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proc. USENIX NSDI*, 2012.

[66] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proc. IEEE ICCV*, 2017.

[67] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, et al. Tensorflow. js: Machine learning for the web and beyond. In *Proc. SysML*, 2019.

[68] Apache Spark. Spark: Dynamic resource allocation. *Spark v2.2.1 Documentation*, 2018.

[69] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *Proc. IEEE INFOCOM*, 2016.

[70] Thomas Stockhammer. Dynamic adaptive streaming over http – standards and design principles. In *Proc. ACM MMSys*, 2011.

[71] Richard S Sutton and Andrew G Barto. *Reinforcement Learning (Second Edition): An Introduction*. MIT press, 2018.

[72] William N Venables and Brian D Ripley. Tree-based methods. In *Modern Applied Statistics with S*, pages 251–269. Springer, 2002.

[73] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *Proc. ICML*, 2018.

[74] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv:1901.00596*, 2019.

[75] Yikai Xiao, Qixia Zhang, Fangming Liu, Jia Wang, Miao Zhao, Zhongxing Zhang, and Jiaxing Zhang. Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning. In *Proc. IEEE/ACM IWQoS*, 2019.

[76] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware internet video delivery. In *Proc. USENIX OSDI*, 2018.

[77] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proc. ACM SIGCOMM*, 2015.

[78] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnn explainer: A tool for post-hoc explanation of graph neural networks. In *Proc. NeurIPS*, 2019.

[79] H. Zhang, L. Song, Y. Li, and G. Y. Li. Hypergraph theory: Applications in 5g heterogeneous ultra-dense networks. *IEEE Communications Magazine*, 55(12):70–76, 2017.

[80] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proc. IEEE CVPR*, 2018.

[81] Ying Zheng, Ziyu Liu, Xinyu You, Yuedong Xu, and Junchen Jiang. Demystifying deep learning in networking. In *Proc. ACM APNet*, 2018.

[82] He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *Proc. ACM PLDI*, 2019.

[83] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Proc. NIPS*, 2010.

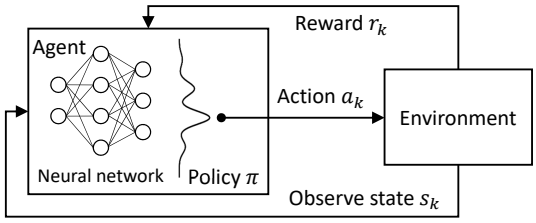


Figure 16: RL with neural networks as policy.

Appendices

A Mathematical Primer

A.1 Graph Neural Network

We briefly introduce the mathematical basics and training methods of GNNs that used in this paper. As introduced above, the key idea of GNNs is *message passing*. This concept can be expressed as follows:

$$M_v = f\left(\{M_u | u \in \text{NBR}(v)\}\right) \tag{13}$$

where NBR(V) represents the neighbor vertices of vertex v, M_v represents the message embedded in vertex v. f is a fine-tuned embedding function, which is usually represented by DNNs. Operators usually need to set initial values for each vertex and repeatedly update the feature values by iteration:

$$M_v^{(t+1)} = f\left(\left\{M_u^{(t)} | u \in \text{NBR}(v)\right\}\right),\ \forall t \geqslant 0 \tag{14}$$

where M_v^t denotes the embedded message at the t-th iteration. M_v^0 is usually initialized as F_v. After several times of iteration, the message value of all vertices will converge to a stable point. In this case, each vertex also contains the information from its neighbor vertices. We refer readers to [74] for a comprehensive understanding of GNN.

A.2 Reinforcement Learning

We briefly introduce the basic knowledge about reinforcement learning used in this paper. We refer the readers to [71] for a more comprehensive understanding of RL.

In RL, given states $\mathscr{S}$, actions $\mathscr{A}$, reward function $R(s)$, transition probabilities $p(s'|s,a)$, and policy: $\pi : \mathscr{S} \to \mathscr{A}$, let:

$$V_t^{(\pi)}(s) = R(s) + \textstyle\sum_{s' \in \mathscr{S}} p\left(s'|s,\pi(s)\right) V_{t+1}^{(\pi)}(s') \tag{15}$$

$$Q_t^{(\pi)}(s,a) = R(s) + \textstyle\sum_{s' \in \mathscr{S}} p\left(s'|s,a\right) V_{t+1}^{(\pi)}(s') \tag{16}$$

be the *value function* and *Q-function* with $V_T^{(\pi)} = 0$ during a finite-horizon (T steps). At each iteration t, the *agent* (flow scheduler) first observers a *state* $s_t \in \mathscr{S}$ from the surrounding *environment*. The agent then takes an *action* $a_t \in \mathscr{A}$ according to its *policy* π. The environment then returns a

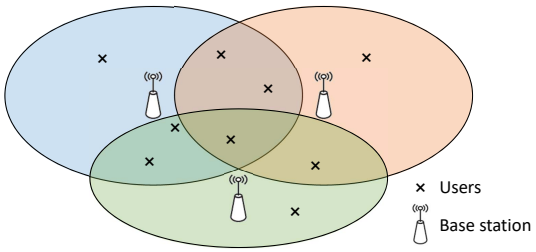


Figure 17: Ultra-dense network.

reward r_t and updates its state to s_{t+1}. Reward is used to indicate how good is the current decision. The goal is to learn a policy π to optimize *accumulated future discounted reward* $\mathbb{E}[\sum_t \gamma^t r_t]$ with the discounting factor $\gamma \in (0,1]$. $\pi_\theta(s,a)$ is the probability of taking action a at state s with policy π_θ parameterized by θ, which is usually represented with DNNs to solve large-scale practical problems [57, 58]. An illustration of RL is presented in Figure 16. See [71] for more details.

B Other Global Control Systems

We present several other application scenarios presented in Table 2 with our hypergraph-based modeling method.

B.1 Cluster Scheduling Model

In cluster scheduling scenarios (e.g., Spark [68]), a job consists of a directed acyclic graph (DAG) whose nodes are the execution stages of the job, as shown in Figure 4(c). A node's task cannot be executed until the tasks from all its parents have finished. The scheduler needs to decide how to allocate limited resources to different jobs [54]. Since the jobs to schedule are naturally graphs, which is a special kind of hypergraph where the degree of hyperedge is two, the cluster scheduling scenario can be naturally integrated into the model provided by TranSys. Vertex features F_V are the work of nodes and hyperedge features F_E are the data transmission between nodes [54].

B.2 Ultra-Dense Cellular Network

In the 5G mobile scenario, one mobile user is usually connected to multiple picocell base stations, which is known as ultra-dense network [42]. Network operators need to decide which base station to connect for each user based on users' traffic demand and base stations' transmission capacity. The scenario could be modeled as a hypergraph [79], with the coverage of the picocell base stations as hyperedges, and mobile users as vertices.

An illustration of hypergraph mapping results is shown in Figure 17. The coverage of each base station is shaded with different colors. Hyperedge features F_E could be the capacity of each base station, etc. Vertex features F_V could be the traffic demand of each mobile user. The traffic optimizer will then continuously select the best base station(s) to connect for each mobile user according to the users' locations.

B.3 NFV Consolidation and Placement

Network function virtualization (NFV) is widely adopted to replace dedicated hardware with virtualized network functions (VNFs). Due to the processing ability and data transmission, one VNF could be replicated to several instances on different servers [31], and multiple VNFs could also be consolidated onto one server [65]. A key problem for network operators is to study where to place their VNF instances. Traditional methods include different heuristics and integer linear programming (ILP). Our observation is that the consolidation and placement problem in NFV could also be modeled with a hypergraph, with servers as hyperedges and VNF as vertices. Hyperedge e connects with vertex v indicates that VNF v has an instance placed onto server e. Hyperedge features F_E could be the processing capacity of servers and vertex features F_V could be the processing speed of different types of VNF.

C Properties of Local Control Systems

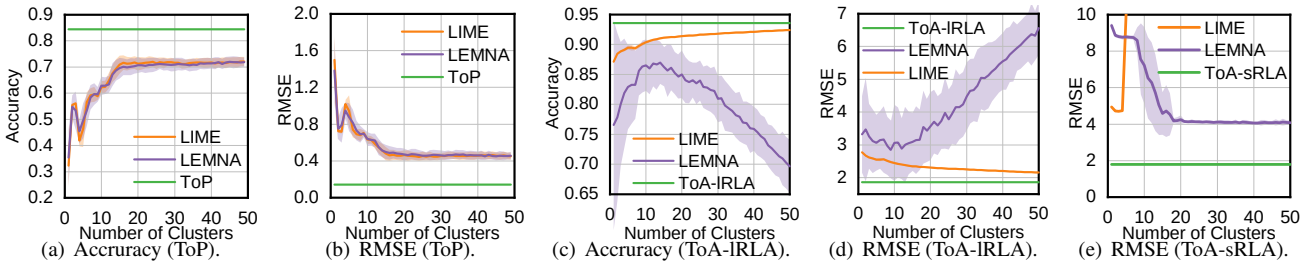
In this section, we present more details on the system-level evaluation of TranSys over LCSes. We compare the performance of TranSys against another two explanation methods in Appendix C.1. We also provide a sensitivity analysis over two hyperparameters in TranSys (number of leaf nodes) in Appendix C.2. We further measure the computation overhead of explaining LCSes with TranSys in Appendix C.3.

C.1 Faithfulness of Single Actions

We further want to know the reason for the performance maintenance of TranSys. We measure the accuracy and root-mean-square error (RMSE) of the decisions made by TranSys compared to the original decisions made by DNNs. As baselines, we compare the faithfulness of TranSys over three DNNs (ToP, ToA-lRLA, ToA-sRLA) with two recent interpretation methods that could provide simpler models:

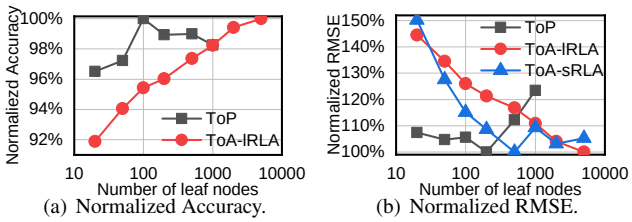
- LIME [61] predicts and explains with linear regression.
- LEMNA [34] employs mixture regression to predict local non-linearity.

As both methods are designed for *local* predictions, to make a fair comparison, we run the baselines in the following way: at training, we first use k-means clustering [50] to cluster the samples into k groups. We then train LIME and LEMNA inside each group. When making a prediction for a new sample, we first find the nearest group to the new sample and apply the results of that group. We vary k from 1 to 50 and repeat the experiments for 100 times to eliminate the randomness during training. Results are shown in Figure 18.



(a) Accuracy (ToP). (b) RMSE (ToP). (c) Accuracy (ToA-lRLA). (d) RMSE (ToA-lRLA). (e) RMSE (ToA-sRLA).

Figure 18: Faithfulness of ToP and ToA. Shadows represent one standard deviation from the average. sRLA predicts real values thus does not have accuracy. Results of LIME over sRLA diverges with $\geqslant$5 clusters.



(a) Normalized Accuracy. (b) Normalized RMSE.

Figure 19: Sensitivity of number leaf nodes on prediction accuracy and RMSE. Results are normalized by the best value on each curve.

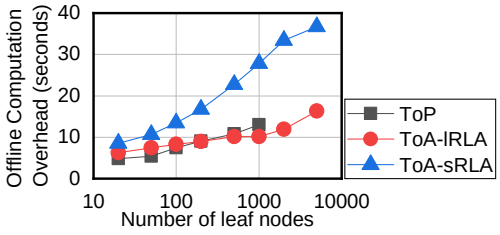


Figure 20: Offline Computation Overhead of TranSys.

ToP and ToA do not need to be clustered; thus, they are constant lines.

From Figure 18(a) and 18(c), ToP and ToA-lRLA respectively achieve high accuracy of 84.3% and 93.6% compared to original DNNs. As the underlying decision logics of state-of-the-art algorithms in flow scheduling [9, 11] are much simpler than those of video bitrate adaption (e.g., stochastic optimization [77], Lyapunov optimization [69]), the accuracy of ToA-lRLA is a little higher than that of ToP. The low decision errors in Figure 18(b), 18(d) and 18(e) indicate that even for those decision tree decisions that are different from DNNs, the error made by TranSys is acceptable, which will not lead to drastic performance degradation. The accurate imitation of original DNNs with decision trees results in the negligible application-level performance loss in §6.2. Meanwhile, the accuracy and RMSE of TranSys are much better than those of LIME and LEMNA. Our design choice in §4.1 is thus verified: decision trees can provide richer expressiveness and are suitable for networked systems. The performance of LEMNA is unstable for two agents of ToA since the states of AuTO is highly centralized at several places from our experiments, which degrades the performance of expectation-maximization iterations in LEMNA [34].

C.2 Sensitivity Analysis.

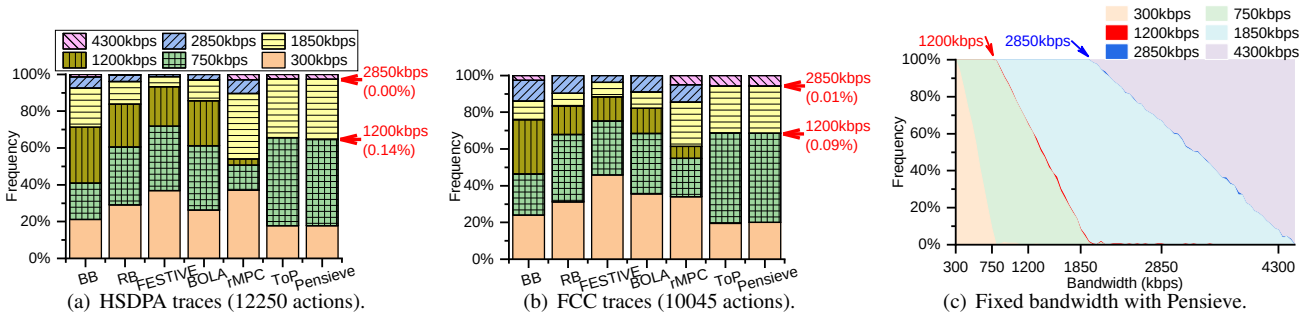
To test the robustness of TranSys against the number of leaf nodes, we vary the number of leaf nodes from 20 to 5000 and measure the accuracy and RMSE for the three agents evaluated in Appendix C.1 (ToP, ToA-sRLA, ToA-lRLA). Results are presented in Figure 19. The accuracy and RMSE of ToP with the number of leaf nodes varying from 20 to 5000 are better than the best results of LIME and LEMNA in Figure 18 in Appendix C.1. ToA-lRLA and ToA-sRLA outperform the best value of LIME and LEMNA in a wide range from 200 to 5000 leaf nodes. This indicates that network operators do not need to spend a lot of time in finetuning the hyper-parameter: a wide range of settings all perform well.

C.3 Computation Overhead

We further examine the computation overhead of TranSys in decision tree extraction. We measure the decision tree computation time of ToP, ToA-lRLA, and ToA-sRLA at different numbers of leaf nodes on our testbed. As the action space of ToP (6 actions) is much smaller than those of ToA-lRLA (108 actions) and ToA-sRLA (real values), the decision tree of ToP has completely been separated with $\sim$1000 leaf nodes. Thus we cannot generate decision trees for ToP with more leaf nodes without enlarging the training set. As shown in Figure 20, even when we set the number of leaf nodes to 5000, the computation time is still less than one minute. Since decision tree extraction is executed offline after DNN training, the additional time is negligible compared to the training time of DNN models (e.g., at least 4 hours in Pensieve with 16 parallel agents [53] and 8 hours in AuTO [18]). TranSys can convert the DNN into decision tree with negligible computation overhead.

D Sensitivity of TranSys over GCSes

We measure the sensitivity of two hyperparameters λ_1 and λ_2 when explaining the GCSes with TranSys as introduced in §5.3. We vary the two λs from 0.125 to 4 and measure the affected term in Equation 6. We measure the $\frac{||W||}{||I||}$ (scale of



(a) HSDPA traces (12250 actions). (b) FCC traces (10045 actions). (c) Fixed bandwidth with Pensieve.

Figure 21: Frequencies of bitrate decisions.

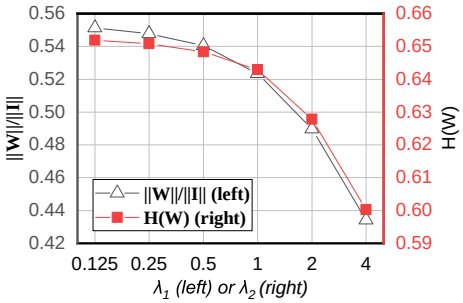


Figure 22: Sensitivity analysis of λ_1 and λ_2. When varing one hyperparameter, the other one is kept unchanged.

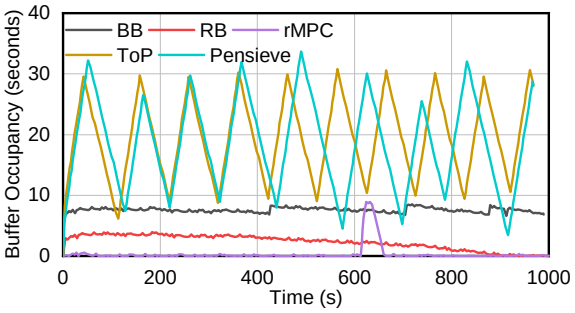


Figure 23: Buffer Occupancy at 3000kbps Link.

W) after training when varying λ_1 and keeping λ_2 unchanged in different experiments and present the results as the black line in Figure 22. We also measure the $H(W)$ (entropy of W) by varying λ_2 and keeping λ_1 unchanged and present the results in red in Figure 22. From the results, we can see that different terms in the optimization goal all actively respond to the changes in respective hyperparameters.

E Troubleshooting Pensieve (Cont'd)

E.1 Observations

When training ToP with the resampled dataset, we observe that Pensieve rarely selects particular bitrates. The frequencies of selected bitrates of the experiments in §6.2 are presented in Figure 21(a) and 21(b). On the one hand, ToP generates almost the same results with Pensieve, which further convince the faithfulness of ToP. On the other hand, 1200kbps and 2850kbps are rarely selected by Pensieve, which raises our interests since they are *intermediate* bitrates. The highest or lowest bitrates may not be selected due to network conditions, but similar reasons do not explain the current observation.

To further explore the reasons, we evaluate Pensieve with the following experiments: We emulate Pensieve on a set of links with fixed bandwidth ranging from 300kbps to 4500kbps and collect the decisions. As the sample video used by [53] is too short to illustrate, we replace the test video with a video of 1000 seconds (251 chunks) and keep all other configurations the same with the original experiment.

As shown in Figure 21(c), 1200kbps and 2850kbps are still not preferred by Pensieve. For example, when the link bandwidth is set around 3000kbps, the optimal decision is constantly selecting 2850kbps, just like 1850kbps is mostly preferred when the bitrate is fixed to around 2000kbps. The bandwidth is a little higher than the bitrate of 2850kbps due to the difference between goodput and link bandwidth. However, in this case, only 0.4% of selections made by Pensieve are 2850kbps, while the remaining decisions are divided between 1850kbps and 4300kbps.

E.2 Reasons Deep Dive

We also provide more details on the experiments of two links with bandwidth fixed to 3000kbps and 1300kbps in §7.1.

3000kbps Link. Except for the experiments in §7.1, we also investigate the runtime buffer occupancy over the 3000kbps link. As shown in Figure 23, the buffer occupancy of Pensieve fluctuates: buffer increases when 1850kbps is selected and decreases when 4300kbps is selected, which is also faithfully mimicked by ToP. The oscillation leads to heavy smoothness penalty. Meanwhile, the buffer occupancy can also explain the poor performance of rMPC in Figure 11: rMPC converges at the beginning thus there is no enough buffer against the fluctuation of chunk size since the size of each video chunk is not exactly the same. Thus heavy rebuffer penalty is imposed on rMPC. The buffer of BB and RB decreases slightly during the total 1000 seconds experiment as the goodput is not exactly 2850kbps (the average bitrate of sample video).

As the raw outputs of the DNNs in Pensieve are the

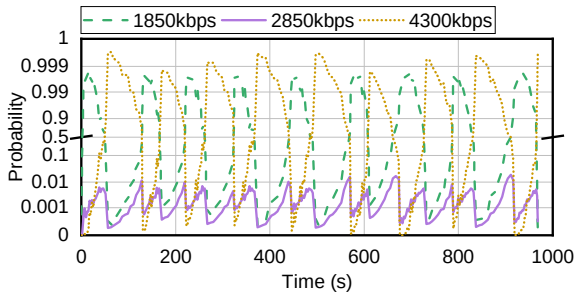
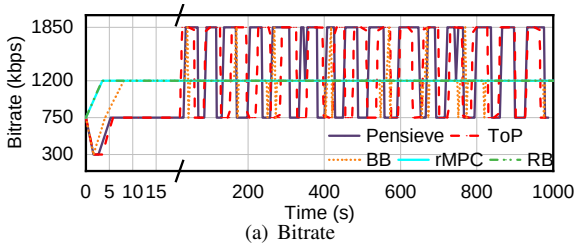
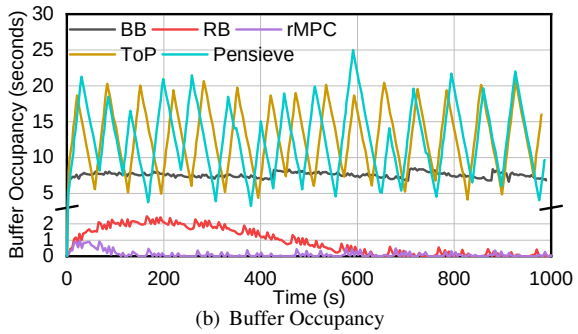


Figure 24: Probabilities of selecting 1850kbps, 2850kbps, 4300kbps qualities. The probability of selecting other three qualities is less than 10^{-4} thus not presented.



(a) Bitrate



(b) Buffer Occupancy

Figure 25: Results on a 1300kbps link.

normalized probabilities of selecting each action, we further investigate the internal probabilities of Pensieve on the 3000kbps link and present the results in Figure 24. Higher probability approaching 1 indicates higher confidence in the decision. We can see that Pensieve does not have enough confidence in the decision it made, which indicates that Pensieve might not experience similar conditions in training; thus, it does not know how to make a decision.

Table 4: QoE on the 1300kbps link.

BB	RB	rMPC	ToP	Pensieve
1.050	0.904	0.803	0.986	0.983

1300kbps Link. We also provide the details about the experiments in Figure 21(c) on a 1300kbps link. Results are shown in Figure 25 and Table 4. The results are similar to the 3000kbps experiment, except that the performance of RB is worse since it converges faster.

F Open Problems

We articulate two critical but still open problems that also prevent learning-based networked systems from practical deployment in real-world scenarios. We call for the efforts from the community to work together towards practical DL-based networked systems.

Dynamic networked systems. Networked systems usually require frequent updates during runtime (e.g., flow table updates in switches). However, since decisions of DNNs are made based on numerous neurons, DL-based systems are known to have difficulty in dynamic adjustments [47]. For example, if we implement the DL-based flow scheduling systems [§7.2] into switches, it's challenging to dynamically adjust the decision boundary of DNN-based scheduling policies like heuristic methods. The transparency enabled by TranSys might somewhat clarify the decision boundary, which, however, still needs human efforts when the number of nodes increases. Employing recent advances in time-changing decision trees [14, 51] at the inference stage could be a potential avenue.

System fairness. The fairness of networked systems has been widely discussed in networked systems, such as cluster job scheduling [32] and congestion control [10]. Network operators should avoid providing unfair services to different users. However, some clues indicate that DNNs are likely to trade a little fairness for a more efficient overall performance [35]. Moreover, since the policies are nontransparent to network operators, it is unknown whether the adoption of DL-based networked systems will impair the fairness among users. Converting DNN-based policies to decision trees might address the fairness problem to some extent by transparentizing the networked systems. Further research with advanced methods (e.g., fair decision trees [6]) is still required to ensure fairness of DL-based networked systems.