Interpreting Deep Learning-Based Networking Systems

Zili Meng
Tsinghua University
zilim@ieee.org

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

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

Mingwei Xu
Tsinghua University
xumw@tsinghua.edu.cn

Hongzi Mao
MIT CSAIL
hongzi@mit.edu

Hongxin Hu
Clemson University
hongxih@clemson.edu

Abstract

While many deep learning (DL)-based networking systems have demonstrated superior performance, the underlying Deep Neural Networks (DNNs) remain blackboxes and stay uninterpretable for network operators. The lack of interpretability makes DL-based networking systems prohibitive to deploy in practice. In this paper, we propose Metis, a framework that provides interpretability for two general categories of networking problems spanning local and global control. Accordingly, Metis introduces two different interpretation methods based on decision tree and hypergraph, where it converts DNN policies to interpretable rule-based controllers and highlight critical components based on analysis over hypergraph. We evaluate Metis over two categories of state-of-the-art DL-based networking systems and show that Metis provides human-readable interpretations while preserving nearly no degradation in performance. We further present four concrete use cases of Metis, showcasing how Metis helps network operators to design, debug, deploy, and ad-hoc adjust DL-based networking systems.

1 Introduction

Recent years have witnessed a steady trend of applying deep learning (DL) to a diverse set of network optimization problems, including video streaming [42, 44, 71], local traffic control [14, 31], parallel job scheduling [36, 45], and network resource management [59, 69, 79]. The key enabler for this trend is the use of Deep Neural Networks (DNNs), thanks to their strong ability to fit complex functions for prediction [38, 39]. Moreover, DNNs are easy to marry with standard optimization techniques such as reinforcement learning (RL) [64] to allow data-driven and automatic performance improvement. Consequently, prior work has demonstrated significant improvement with DNNs over hand-crafted heuristics in multiple network applications [14, 44, 45].

However, the superior performance of DNNs comes at the cost of using millions or even billions of parameters [38, 51]. This cost is fundamentally rooted in the design of DNNs, as they typically require numerous parameters to achieve universal function approximation [39]. Therefore, network operators have to consider DNNs as large blackboxes [18, 82], which makes DL-based networking systems incomprehensible to debug, heavyweight to deploy, and extremely difficult to ad-hoc adjust (§2.1). As a result, network operators firmly hold a general fear against using DL-based networking systems for critical deployment in practice.

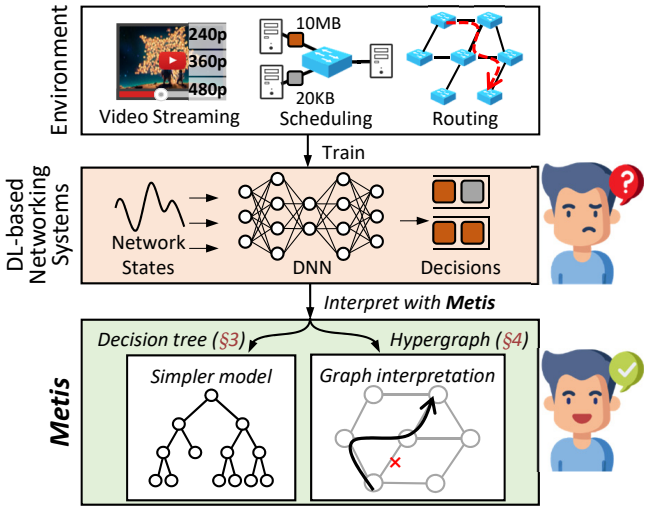


Figure 1: High-level workflow of Metis.

Over the years, the machine learning community has developed several techniques for understanding the behaviors of DNNs in the scope of image recognition [7, 76] and language translation [56, 65]. These techniques focus on surgically monitoring the activation of neurons to determine the set of features that the neurons are sensitive to [7]. However, directly applying these techniques to DL-based networking systems is not suitable — network operators typically seek simple, deterministic control rules mapped from the input (e.g., scheduling packets with certain headers to a port), as opposed to nitpicking the operational details of DNNs. Besides, networking systems are diverse in terms of their application settings (e.g., distributed control v.s. centralized decision making) and their input data structure (e.g., time-series of throughput and routing paths in a topology). The current DNN interpretation tools, designed primarily for well-structured vector inputs (e.g., images, sentences), are not sufficient across diverse networking systems. Therefore, an interpretable DL framework specifically tailored for the networking domain is much needed.

In this paper, our high-level design goal is to interpret DL-based networking systems with human-readable control

Category	Scenario	Examples
Local	End-based congestion control	Aurora [31]
	Client-based video streaming	Pensieve [44]
	On-switch flow scheduling	AuTO [14]
Global	Cluster job scheduling	Decima [45]
	SDN routing optimization	RouteNet [59]
	Network function (NF) placement	NFVdeep [69]

Table 1: Local systems collect information and make decisions locally (e.g., from end-devices or switches only). Global systems aggregate information and make decisions across the network.

policies so that network operators can easily debug, deploy, and ad-hoc adjust DL-based networking systems. We develop Metis[1], a general framework that contains an ensemble of techniques to provide interpretability. To support a wide range of networking systems, Metis leverages an abstraction that separates current networking systems into *local systems* and *global systems* (Figure 1). In this separation, local systems collect information locally and make decisions for one instance only, such as congestion control agents on end-devices and flow schedulers on switches. By contrast, global systems aggregate information across the network and make global planning for multiple instances, such as the controller in a software-defined network (SDN). Table 1 presents typical examples that fall into these two categories. For each category, Metis uses different techniques to achieve interpretability, as depicted in Figure 2.

Specifically, we adopt a decision tree conversion method [6, 58] for local systems. The main observation behind the design choice is that existing heuristic local systems are usually *rule-based* decision-making systems (§3.1). The conversion is built atop a teacher-student training process, where the DNN policy acts as the teacher and generates input-output samples to construct the student decision tree [58]. However, to match the performance with DNNs, traditional decision tree algorithms [24] usually output an exceedingly large number of branches, which are effectively uninterpretable. We leverage two important observations to prune the branches down to a tractable number for network operators. First, sensible policies in local systems often unanimously output the same control action for a large part of the observed states. For example, any performant bitrate adaption (ABR) policies [44] would keep a low bitrate when both of the bandwidth and the playback buffer are low. By relying on the data generated by the teacher DNN, the decision tree can easily cut down the decision space. Second, different input-output pairs have different contributions to the performance of a policy. We adopt a special resampling method [6] that allows the teacher DNN to guide the decision tree to prioritize the actions leading to the best outcome. Empirically, our decision tree can generate human-readable interpretations (§6.1), and the performance degradation is within 2% of the original DNNs (§6.4).

For global systems, our observation is that we can formulate many of them with hypergraphs. The reason behind the observation is that most global systems either have

[1]Metis is a Greek deity that offers wisdom and consultation.

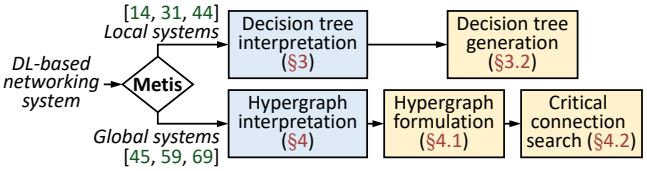


Figure 2: Metis introduces different interpretation methods for local and global DL-based networking systems.

graph-structured inputs or construct a mapping between two variables, both of which could be formulated with hypergraphs (§4.1). For example, given routing results of a DL-based routing optimizer [59], we can formulate the interaction between routing paths and links as the relationship between hyperedges[2] and vertices. The placement of network functions (NFs) [69] could also be formulated as a hypergraph, where NFs and physical servers are hyperedges and vertices, and the placement algorithm constructs a mapping between them. With hypergraph formulations, Metis computes the importance of each part of the hypergraph by constructing an optimization problem (e.g., finding critical routing decisions to the overall performance) (§4.2). With the importance of each decision, network operators can interpret the behaviors of DL-based networking systems (§6.1).

For concrete evaluation, we generate interpretable policies for two types of DL-based networking systems with Metis (§6.1). For example, we interpret the bitrate adaptation policy of Pensieve [44] and recommend a new decision variable. We also present four use cases of Metis in the design, debugging, deployment, and ad-hoc adjustment of DL-based networking systems. (i) Metis helps network operators to redesign the DNN structure of Pensieve with a quality of experience (QoE) improvement by 5.1%[3] on average (§6.2). (ii) Metis debugs the DNN in Pensieve and improves the average QoE by up to 4% with only decision trees (§6.3). (iii) Metis enables a lightweight DL-based flow scheduler (AuTO [14]) and a lightweight Pensieve with shorter decision latency by 27× and lower resource consumption by up to 156× (§6.4). (iv) Metis helps network operators to adjust the routing paths of a DL-based routing optimizer (RouteNet [59]) when ad-hoc adjustments are needed (§6.5).

We make the following contributions in this paper:

- Metis, a framework to provide interpretation for two general categories of DL-based networking systems, where it interprets local systems with decision trees (§3) and global systems with hypergraphs (§4).
- Prototype implementations of Metis over three DL-based networking systems (Pensieve [44], AuTO [14], and RouteNet [59]) (§5), and their interpretations with capturing well-known heuristics and discovering new knowledge (§6.1).
- Four use cases on how Metis can help network operators to design (§6.2), debug (§6.3), deploy (§6.4), and ad-hoc adjust (§6.5) DL-based networking systems.

[2]Similar to an edge connecting two vertices in a graph, a hyperedge covers multiple vertices in the hypergraph (§4.1).

[3]Even a 1% improvement in QoE is significant to current Internet video providers (e.g., YouTube) considering the volume of videos [42].

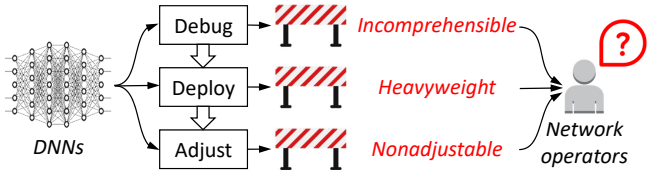


Figure 3: DNNs create barriers for network operators in many stages of the development flow of networking systems.

To the best of our knowledge, Metis is the first general framework to interpret a diverse set of DL-based networking systems at deployment. The Metis project and our use cases are available at https://metis.transys.io. We believe that Metis will accelerate the deployment of DL-based networking systems in practice.

2 Motivation

We motivate the design of Metis by analyzing (i) the drawbacks of current DL-based networking systems (§2.1), and (ii) why existing interpretation methods are insufficient for DL-based networking systems (§2.2).

2.1 Drawbacks of Current Systems

The blackbox property of DNNs lacks interpretability for network operators. Without understanding why DNNs make decisions, network operators might not have enough confidence to adopt them in practice [82]. Moreover, as shown in Figure 3, the blackbox property brings drawbacks to networking systems in debugging, online deployment, and ad-hoc adjustment due to the following reasons.

Incomprehensible structure. DNNs could contain thousands to billions of neurons [51], making them incomprehensible for human network operators. Due to the complex structure of DNN, when DL-based networking systems fail to perform as expected, network operators will have difficulty in locating the erroneous component. Even after finding the sub-optimality in the design of DNN structures, network operators are challenged to redesign them for better performance. If network operators could trace the mapping function between inputs and outputs, it would be easier to debug and improve DL-based networking systems.

Heavyweight to deploy. DNNs are known to be bulky on both resource consumption and decision latency [30]. Even with advanced hardware (e.g., GPU), DNNs may take tens of milliseconds for decision-making (§6.4). In contrast, networking systems, especially local systems on end devices (e.g., mobile phones) or in-network devices (e.g., switches), are resource-limited and latency-sensitive [30]. For example, loading a DNN-based ABR algorithm on mobile clients increases the page load time by around 10 seconds (§6.4), which will make users leave the page. Existing systems usually provide "best-effort" services only and roll back to heuristics when resource and latency constraints can not be met [14], which degrades the performance of DNNs.

Nonadjustable policies. Practical deployment of networking systems also requires ad-hoc adjustments or adding temporary features. For example, we could adjust the weights for different jobs in fair scheduling to catch up with the fluctuations in workloads [45]. However, the lack of interpretation brings difficulties to network operators when they need to adjust the networking systems. Without understanding why DNNs make such decisions, arbitrary adjustments may lead to severe performance degradation. For example, when network operators want to manually reroute a flow away from a link, without interpretations of decisions, network operators might not know how and where to accommodate that flow.

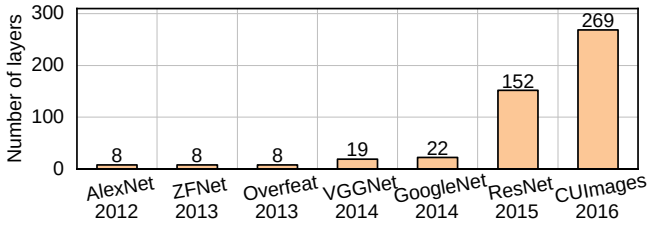


Figure 4: The exponential growth of DNN complexity in ImageNet Challenge winners [17] (Figure adopted from [20]).

Discussions. The application of DNNs in networking systems is still at a preliminary stage: DNNs of Pensieve [44], AuTO [14], and RouteNet [59] (published in 2017, 2018, and 2019) all have less than ten layers. As a comparison, a sharp increase in the number of DNN layers has been observed in other communities (Figure 4). Recent language translation models even contain billions of parameters [51]. Although we are not saying that the larger is the better, it is indisputable that larger DNNs will aggravate the problems and create barriers to deploy DL-based networking systems in practice.

2.2 Why Not Existing Interpretations?

For DL-based networking systems, existing interpretation methods [19, 26] are insufficient in the following aspects:

Different interpretation goal. The question of *why a DNN makes a certain decision* may have answers from two angles. In the machine learning community, the answer could be understanding the *inner mechanism of ML models* (e.g., which neurons are activated for some particular input features) [7, 76]. It's like trying to understand how the brain works with surgery. In contrast, the expected answer from network operators is the *relationship between inputs and outputs* (e.g., which input features affect the decision) [82]. What network operators need is a method to interpret the mapping between the input and output for DNNs.

Diverse networking systems. As shown in Table 1, DL-based networking systems have different application scenarios and are based on various DL approaches, such as feedforward neural network (FNN) [44], recurrent neural network (RNN) [72], and graph neural network (GNN) [45]. Therefore, interpreting diverse DL-based networking systems with one single interpretation method is insufficient. For example, LEMNA [27] could only interpret the behaviors of RNN and thus is not suitable for GNN-based networking systems [45]. In Metis, we observe that DL-based networking systems can be divided into two categories (local and global) and develop corresponding techniques for each category.

Non-standard state and action spaces. Existing interpretation methods are usually designed with easy-encoded state and action spaces. For example, methods interpreting image classification tasks are designed for the grid-based RGB encoding [7, 76]. The interpretation methods for language translation tasks are also based on vectorized word embeddings [56, 65]. However, networking systems inherently work with non-standard state and action spaces. For example, RouteNet [59] takes the topology as input and generates variable-length routing paths. Specially designed interpretation methods for networking systems are hence needed.

In response, to interpret DL-based networking systems, Metis introduces a decision tree-based method together with a hypergraph-based method for different categories of systems. Our observation is that although DL-based networking systems are diverse, when divided into two categories (local and global), the commonality inside each category enables us to design specific interpretation methods.

3 Decision Tree Interpretations

In this section, we first describe the design choice for choosing decision trees for local systems in Metis (§3.1), and then explain the detailed methodology to convert the DNNs to decision trees (§3.2).

3.1 Design Choice: Decision Tree

As introduced in §1, Metis converts DNNs into simpler models based on interpretation methods. There are many candidate models, such as (super)linear regression [27, 55], decision trees [6, 58], etc. We refer the readers to [19, 26] for a comprehensive review.

In this paper, we decide to convert DNNs to *decision trees* due to three reasons. First, the logic structure of decision trees resembles the policies made by networking systems, which are rule-based policies. For example, flow scheduling algorithms on switches usually depend on a set of forwarding rules, such as shortest-job-first [5]. ABR algorithms depend on precomputed rules over buffer occupancy and predicted throughput [63, 73]. Second, decision trees have rich expressiveness and high faithfulness because they are non-parametric and can represent very complex policies [9]. We demonstrate the performance of decision trees during conversion compared to other methods [27, 55] in Appendix E. Third, decision trees are lightweight for networking systems, which will bring further benefits to resource consumption and decision latency (§6.4). There are also research efforts that interpret DNNs with programming language [67, 84]. However, designing different primitives for each networking system is time-consuming and inefficient.

With interpretations of local systems in the form of decision trees, we can interpret the results since the decision-making process is transparent (§6.1). Also, we can debug the DNN models when they generate sub-optimal decisions (§6.3). Furthermore, since decision trees are much smaller in size, less expensive on computation, we could also deploy the decision trees online instead of deploying heavyweight DNN models. This will result in low decision-making latency and resource consumption (§6.4).

3.2 Conversion Methodology

To extract the decision tree from a trained DNN, we adopt a teacher-student training methodology proposed by Bastani et al. [6]. We tailor the approach for networking systems with key conversion steps as follows:

Step 1: Traces collection. When training decision trees, it is important to obtain an appropriate dataset from DNNs. Simply covering all possible (state, action) pairs is too costly and does not faithfully reflect the state distribution from the target policy. Thus, Metis follows the trajectories generated by the teacher DNNs. Moreover, networking systems are sequential decision-making processes, where each action has long-lasting effects on future states. Therefore, the decision tree can deviate significantly from the trajectories of DNNs due to imperfect conversion [6]. To make the converted policy more robust, we let the DNN policy take over the control on the deviated trajectory and re-collect (state, action) pair to refine the conversion training. We iterate the process until the deviation is confined (i.e., the converted policy closely tracks the DNN trajectory).

Step 2: Resampling. Local systems usually optimize *policies* instead of independent actions [14, 31, 44]. In this case, different actions of networking systems may have different importance to the optimization goal. For example, an ABR algorithm downloading a huge chunk at extremely low buffer will lead to a long stall, resulting in severe performance degradation. Meanwhile, downloading a little larger chunk when network condition and buffer are moderate will not have drastic effects. However, decision tree algorithms are designed to optimize the accuracy of a single action and treat all actions the same. Therefore, their optimization goals do not match. Existing DL-based local systems adopt reinforcement learning (RL) to optimize the policy instead of single actions, where the *advantage* of each (state, action) represents the importance to the optimization goal. Therefore, we follow recent advances in converting DNNs in RL policies into decision trees [6] and resample $\mathcal{D}$ according to the advantage function. For each pair (s, a), the sampling probability $p(s, a)$ could be expressed as:

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

where $V(s)$ and $Q(s, a)$ are the value function and Q-function of RL [64]. Value function represents the expected total reward starting at state s and following the policy π. Q-function further specifies the next step action a. π^* is the DNN policy, and A is the action space. $\mathbb{1}(x)$ is the indicator function, which equals to 1 if and only if x is true. We analyze Equation 1 with more details in Appendix A. We then retrain the decision tree on the resampled dataset. Our empirical results demonstrate that the resampling step can improve the QoE over 73% of the traces (Appendix A).

	Scenario	Vertex	Hyperedge	Meaning of $I_{ev} = 1$	Details
#1	SDN routing optimization	Physical link	Path (src-dst pairs)	Path e contains link v.	§ 4.1
#2	Network function placement	Physical server	Network function	One instance of NF e is on server v.	Appendix B.1
#3	Ultra-dense cellular network	Mobile user	Base station coverage	Base station e covers user v.	Appendix B.2
#4	Cluster job scheduling	Job node	Dependency	Dependency e is related to node v.	Appendix B.3

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

Step 3: Pruning. As the size of the decision tree sometimes becomes much larger than network operators can understand, we adopt cost complexity pruning (CCP) [24] to reduce the number of branches according to the requirements from network operators. Compared with other pruning methods, CCP empirically achieves a smaller decision tree with a similar error rate [47]. At its core, CCP creates a cost function of the complexity of the pruned decision tree to balance between accuracy and complexity. Moreover, for the continuous outputs in networking systems (e.g., queue thresholds [14]), we employ the design of the *regression tree* to generate real value outputs [66]. In our experiments, for Pensieve, the size of leaf nodes may be up to 1000 without pruning (Appendix F). With CCP, pruning the decision tree down to 200 leaf nodes only results in a performance degradation of less than 0.6% (§6.4).

Step 4: Deployment. Finally, network operators could deploy the converted model online and enjoy both the performance improvement brought by deep learning and the interpretability provided by the converted model. Our evaluation shows that the performance degradation of decision trees is less than 2% for two DL-based networking systems (§6.4). We also present further benefits of converting DNNs of networking systems into decision trees (easy debugging and lightweight deployment) in §6.3 and §6.4.

4 Hypergraph Interpretations

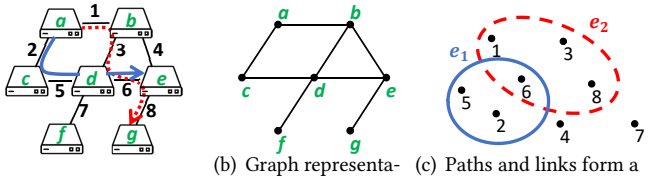
We first briefly introduce hypergraph and present several applications on how to formulate networking systems with hypergraphs (§4.1), and then introduce our interpretation methods to find critical components in hypergraphs (§4.2).

4.1 Hypergraph Formulation

A hypergraph is composed of *vertices* and *hyperedges*. The main difference between the edge in a graph and the hyperedge in a hypergraph is that a hyperedge can cover *multiple* vertices, as shown in Figures 5(b) and 5(c). We denote the set of all vertices and all hyperedges as $\mathcal{V}$ and $\mathcal{E}$. Each vertex v and hyperedge e may also attach their features, denoted as f_v and f_e. We denote the matrix of features of all vertices and hyperedges as F_V and F_E, respectively.

With hypergraph, we can formulate many global systems uniformly, as shown in Table 2. In the following, we will introduce the formulation of SDN routing optimization (scenario #1) in detail and leave other formulations in Appendix B.

Case study: SDN routing optimization. We first present a case study of formulating SDN routing with hypergraph. The SDN controller collects the information from all data plane switches. In this case, an SDN routing optimizer analyzes the traffic demands for each src-dst pair and generates the



(a) Physical topology. (b) Graph representation of Figure 5(a). (c) Paths and links form a hypergraph.

Figure 5: The hypergraph representation of the SDN routing model. Hypergraph could efficiently represent path information.

routing paths for all src-dst traffic demands based on the topology structure and link capacity. However, composed of variant-length switches and links, routing paths are high-order information and are difficult to be efficiently expressed. Previous research efforts try to represent the paths with integer programming [77], which is hard to be efficiently optimized within a limited time. RouteNet [59] designs a DNN-based optimization algorithm to continuously select the best routing paths for each src-dst traffic demand pair.

To formulate the system with hypergraph, we consider the paths as hyperedges and physical links as vertices. A hyperedge covering a vertex indicates the path of that pair of demand contains a link. An illustration of hypergraph mapping results is shown in Figure 5. Links $(1, 2, \cdots, 8)$ are modeled as vertices. Two pairs of transmission demand ($a \Rightarrow e$ and $a \Rightarrow g$) are modeled as hyperedges (denoted as e_1 and e_2). Vertex features F_V are the link capacity. Hyperedge features F_E are the traffic demand volume between each pair of switches. If a hyperedge e covers vertex v, the respective flow of e should go through the respective link of v.

RouteNet generates the overall routing results, i.e., the path of all traffic demands. For example, assume that RouteNet decides the demand from a to e going through link 2, 5, 6 (path in blue), and the demand from a to g going through link 1, 3, 6, 8, the respective hypergraph should be Figure 5(c). Hyperedge e_1 covers vertices 2, 5, 6, and hyperedge e_2 covers 1, 3, 6, 8. All vertex-hyperedge connections $\{(v, e)\}$ are:

$$\{(2, e_1), (5, e_1), (6, e_1), (1, e_2), (3, e_2), (6, e_2), (8, e_2)\} \quad (2)$$

Later in §4.2, we are going to find out which connections are critical to the overall routing decisions of the topology.

Capability of hypergraph representation. We empirically summarize two key features that enable global systems to be formulated with hypergraph:

- *Graph-structured inputs or outputs.* Since a graph is a simple form of a hypergraph, if the inputs or outputs of a global system are graph-structured (e.g., network topology [59], dataflow computation graph [45]), this system can be naturally formulated with hypergraph.

$$\min \ell(W) \quad \text{s.t. } 0 \leqslant W_{ev} \leqslant I_{ev},\ \forall v \in \mathcal{V}, e \in \mathcal{E} \tag{4}$$

where

$$\ell(W) = D(Y_W, Y_I) + \lambda_1 ||W|| + \lambda_2 H(W) \tag{5}$$

$$D(Y_W, Y_I) = \begin{cases} \sum Y_W \log \frac{Y_W}{Y_I} & \text{(discrete)} \\ \sum ||Y_W - Y_I||^2 & \text{(continuous)} \end{cases} \tag{6}$$

$$||W|| = \sum_{v,e} |W_{ev}| \tag{7}$$

$$H(W) = -\sum_{v,e} (W_{ev} \log W_{ev} + (1 - W_{ev}) \log(1 - W_{ev})) \tag{8}$$

Figure 6: Formulation of critical connection search optimization.

- *Bivariate mapping*. If a global system constructs a mapping between two variables, those two variables could be formulated with vertices and hyperedges. The mapping could be formulated the connection relationship in the hypergraph. Many resource allocation systems construct the mapping between resources (e.g., physical servers) and requests (e.g., network functions) [69].

As long as a global system has one of the features above, we can formulate it with hypergraphs and interpret it with Metis. Our observation is that many global systems have at least one feature. For example, in Table 2, scenario #1 processes network topology and scenario #4 processes dataflow graph, both of which are graph-structured. Scenario #2 maps the NF instances to servers and scenario #3 maps each mobile user to a base station, both of which are bivariate mappings.

4.2 Critical Connections Search

Next, we are going to find out which vertex-hyperedge connections are critical to the optimization result of the original system. We first introduce the *incidence matrix* representation of a hypergraph. Incidence matrix I (with the size of $|\mathcal{E}| \times |\mathcal{V}|$) is a 0-1 matrix to represent the connection relationship between vertices and hyperedges. $I_{ev} = 1$ indicates hyperedge e contains vertex v. For example, the incidence matrix of the hypergraph in Figure 5(c) is:

$$I = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{3}$$

Our design goal is to evaluate how each connection is critical to the optimization results of the original system. Taking the case of SDN routing as an example, Metis is going to evaluate how each (link, path) connection in Equation 2 is critical to the overall routing result. We allow a fractional incidence matrix $W \in [0,1]^{|\mathcal{E}| \times |\mathcal{V}|}$ to represent the significance of each hyperedge-vertex connection. $W_{ev} = 0$ if there is no connection between v and e. We first present the overview of the critical connection searching algorithm in Figure 6. The optimization objective in Equation 4 consists of the following three parts:

Performance degradation ($\mathbf{D(Y_W, Y_I)}$). The critical connections should be those connections that have a great influence on the output of the networking system, which is task-independent. Therefore, we need to measure the output of the original DL-based networking system when input features of hyperedges and vertices are weighted by the mask W. Taking the SDN routing case in §4.1 as an example, routing decisions generated by the masked features (demands, capacities) should be similar to the original ones. We denote the decisions generated by the original inputs and inputs with mask W as Y_I and Y_W. Thus, we maximize the similarity between the Y_W and Y_I, denoted as $D(Y_W, Y_I)$ in Equation 5. We adopt KL-divergence [37] to measure discrete outputs (e.g., sequences of routing decisions) and mean square error for continuous outputs, both of which are common similarity metrics in the DL community [35], as shown in Equation 6.

Interpretation conciseness ($||\mathbf{W}||$). Usually, the number of interpretations that humans can understand is budgeted [55]. Therefore, the number of critical connections should also be concise enough to be understandable for network operators. If the algorithm provides too many "critical" connections, network operators will be confused and cannot easily interpret the networking systems. In Metis, we measure the conciseness of W as the sum of all elements (the scale of the matrix). We also need to penalize the scale of mask W in the optimization goal, as shown in Equation 7.

Determinism (H(W)). Moreover, we also expect the results of W to be *deterministic*, i.e., for each connection (v, e), it is either seriously suppressed (W_{ev} close to 0) or almost unaffected (W_{ev} close to 1). Otherwise, the crafty agent will learn to mask all connections with the same weight and generate meaningless interpretations. In this paper, Metis optimizes the *entropy* of mask W to encourage the connections in W to be close to 1 or 0, where the entropy is a measure of uncertainty in the information theory [60], as shown in Equation 8.

To balance the optimization goals above, we provide two customizable hyperparameters (λ_1 and λ_2) for network opreators due to the differences in operators' understandability and application scenarios of systems. For example, an online monitor of routing results may only need the most critical information for fast decisions, while an offline analyzer of routing results requires more detailed interpretations for further improvement. In this case, network operators can increase (or decrease) λ_2 to reduce (or increase) the number of undetermined connections with median mask values. Metis will then expose less (or more) critical connections to network operators. We empirically study the effects of setting λ_1 and λ_2 for network operators in Appendix F.2.

In this way, we can quantitatively know how critical the connection contributes to the output. In the SDN routing case, instead of trivially identifying links where many flows run through, Metis can provide finer-grained interpretations by further identifying *which flow on which link* plays a dominant role in the overall result. We present the interpretations and further improvements in this case in §6.1 and §6.5.

5 Implementation

We interpret two local systems, Pensieve [44] and AuTO [14], and one global system, RouteNet [59], with Metis. Parameter and testbed settings are introduced in Appendix C.

Pensieve implementation. In current Internet video transmissions, each video consists of many *chunks* (a few seconds of playtime), and each chunk is encoded at multiple bitrates [44]. 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. 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 [57] and 205 FCC traces [1]. We integrate DNNs into JavaScript with `tf.js` [61] to run Pensieve in the browser. We set up the same environment and QoE metric with Pensieve.

We then implement Metis+Pensieve. We use the finetuned model provided by [44] to generate the decision tree. We use five baseline ABRs (BB [29], RB [44], Festive [32], BOLA [63], rMPC [73]) as Pensieve and migrate them into `dash.js` [2].

AuTO implementation. AuTO is a flow scheduling system to optimize flow completion time (FCT) based on deep RL. Limited by the long decision latency of DNN, AuTO can only optimize long flows individually with a long-flow RL agent (lRLA). For short flows, AuTO makes decisions locally with multi-level feedback queues [5] and optimizes the queue thresholds with a short-flow RL agent (sRLA). lRLA takes {5-tuples, priorities} of running long flows, and {5-tuples, FCTs, flow sizes} of finished long flows as states and decides the {priority, rate limit, routing path} for each running 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: web search (WS) traces [25] and data mining (DM) traces [3]. We train the DNNs following the instructions in [14]. All other configurations (e.g., link capacity, link load, DNN structure) are set the same as AuTO. We then evaluate the decision tree generated by Metis (Metis+AuTO).

RouteNet* implementation. We train the model on the traffic dataset of the NSFNet topology provided by RouteNet, as presented in Figure 8. We adopt the close-loop routing system in RouteNet, denoted as RouteNet*, which concatenates latency predictions with routing decisions.

As for Metis+RouteNet*, to implement the constraint of W in Equation 4, we adopt the *gating* mechanism used in the machine learning community [16]. Specifically, the incidence matrix value I_{ve} acts as a gate to bound the mask value W_{ve}. We construct a matrix $W' \in \mathbb{R}^{|E|\times|V|}$ and get mask matrix W by the following equation:

$$W = I \circ \text{sigmoid}(W') \tag{9}$$

$\circ$ means element-wise multiplication, and sigmoid function is applied to each element separately. Since the output of sigmoid function is limited in $(0, 1)$, W_{ve} will always be less than or equal to I_{ve}. In this case, the constraint in Equation 4 will be followed during the optimization.

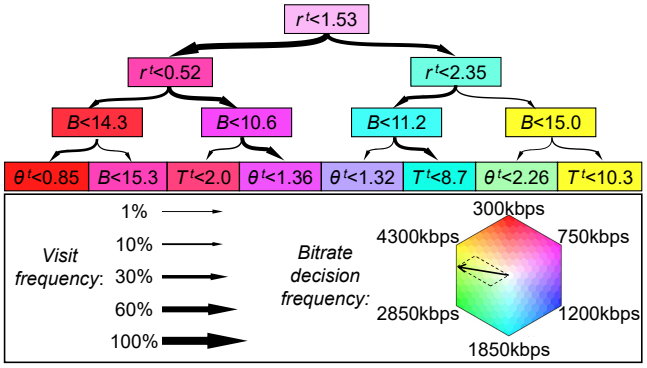


Figure 7: Top 4 layers of the decision tree of Metis+Pensieve. The color represents the frequency of bitrate selections at that node. For example, the arrow in the palette represents that 67% states traversing a node with that color are finally decided as 4300kbps, and 33% states are 2850kbps. Better viewed with color.

6 Experiments

In this section, we first empirically evaluate the interpretability of Metis with two types of DL-based networking systems. Subsequently, we showcase how Metis addresses the drawbacks of existing DL-based networking systems (§2.1). We finally benchmark the interpretability of Metis. Overall, our experiments cover the following aspects:

- **System interpretations.** We demonstrate the effectiveness of Metis by presenting the interpretations of one local system (Pensieve) and one global system (RouteNet*) with newly discovered knowledge (§6.1).
- **Guide for model design.** We present a case on how to improve the DNN structure of Pensieve for better performance based on the interpretations of Metis (§6.2).
- **Enabling debuggability.** With a use case of Pensieve, Metis debugs a problem and improves its performance by adjusting the structure of decision trees (§6.3).
- **Lightweight deployment.** For local systems (AuTO and Pensieve), network operators could directly deploy the converted decision trees provided by Metis online and achieve benefits enabled by lightweight deployments (§6.4).
- **Ad-hoc adjustments.** We provide a case study on how network operators can adjust the decision results of RouteNet* based on the interpretations provided by Metis (§6.5).
- **Metis deep dive.** We finally evaluate the interpretation performance, parameter sensitivity, and computation overhead of Metis under different settings (§6.6).

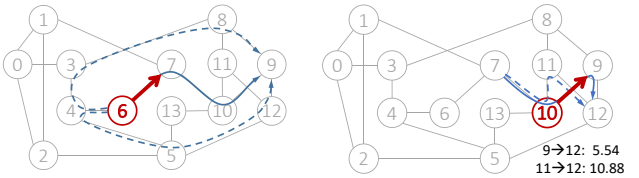
6.1 System Interpretations

With Metis, we interpret the DNN policy learned by a local system, Pensieve, and a global system, RouteNet*.

Local system interpretations. We present the top 4 layers of the decision tree of Metis+Pensieve in Figure 7. The decision variables of each node include the last chunk bitrate (r^t, circle), previous throughput (θ^t, diamond), buffer

	Routing path	Link	Mask M_{ve}	Interpretation type
#1	6⟶7⟶10⟶9	6⟶7	0.886	Shorter
#2	1⟶7⟶10⟶9	1⟶7	0.880	Shorter
#3	7⟶10⟶9⟶12	10⟶9	0.878	Less congested
#4	8⟶3⟶0⟶2	8⟶3	0.875	Shorter
#5	6⟶4⟶3⟶0	6⟶4	0.874	Less congested

Table 3: Top 5 mask value interpretations in Figure 8.



(a) Connection #1. (b) Connection #3. Traffic load of two links are in the bottom-right corner.

Figure 8: Solid blue paths are the results generated by RouteNet. Dashed paths are candidates serving the same src-dst demand. Critical decisions interpreted by Metis are colored red.

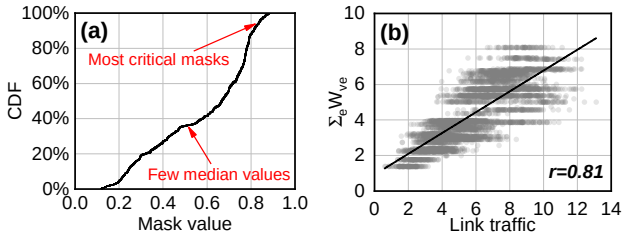
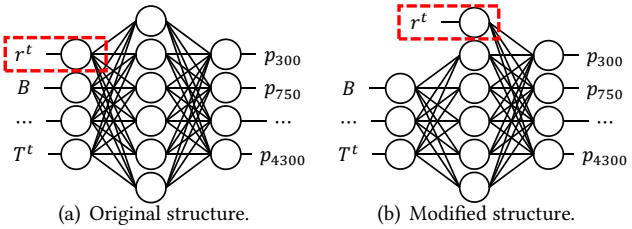


Figure 9: (a) The distribution of mask values in 50 experiments. (b) Sum of mask values ($\sum_e W_{v,e}$) is correlated to the link traffic.

occupancy (B, square), and last chunk download time (T_t, triangle). Since we only present the top 4 layers of the decision tree, we represent the frequency of final decisions of each node with the color on the palette in Figure 7.

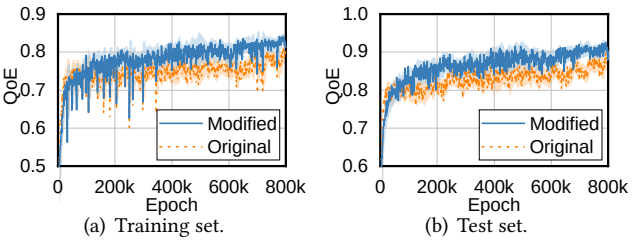
From the interpretations in Figure 7, we can know the reasons behind the superior performance of Pensieve in two directions. (i) *Discovering new knowledge.* On the top two layers, Metis+Pensieve first classifies inputs into four branches based on the *last chunk bitrate*, which is different from existing methods. The information contained in the last bitrate choice affects the output QoE significantly. Based on this observation, we recommend that network operators could design new ABR algorithms with particular focus on the last chunk bitrate. We present a use case on how to utilize this observation to improve the DNN structure in §6.2. (ii) *Capturing existing heuristics.* After that, similar to existing methods, Metis+Pensieve makes decisions based on buffer occupancy (B) [29, 62, 63] and predicted throughput (θ^t and T^t) [2, 73]. With the interpretations provided by Metis, network operators can understand how Pensieve makes decisions.

Global system interpretations. We interpret RouteNet* with Metis and present the top-5 mask values in Table 3. For each path-link connection, there are two common reasons behind selecting path a instead of path b. (i) *Path a is shorter than path b.* For example, connection #1 in Table 3 (path 6⟶7⟶10⟶9 + link 6⟶7) has a high mask value, indicating



(a) Original structure. (b) Modified structure.

Figure 10: We modify the DNN structure of Pensieve based on the interpretations in §6.1. Although two structures are equivalent for the expressive ability, putting significant inputs near to the output will make the DNN optimize easier and better.



(a) Training set. (b) Test set.

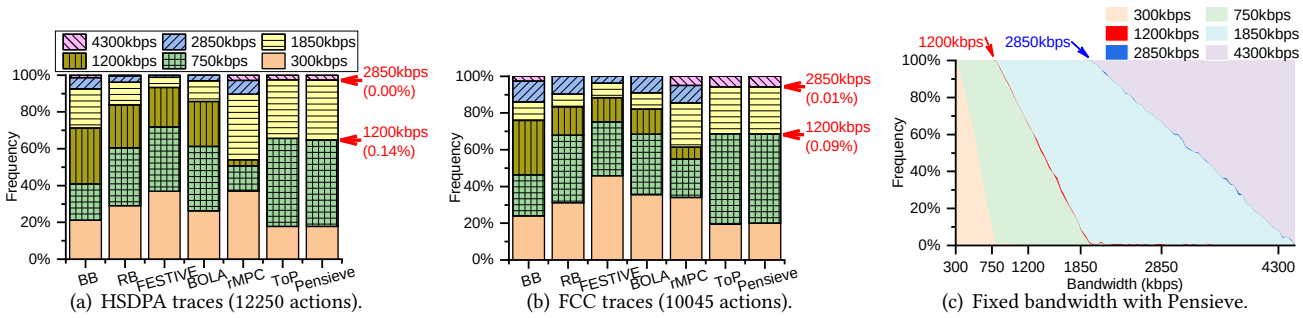
Figure 11: The modification in Figure 10 could improve both the QoE and the training efficiency. Shaded area spans ± std.

selecting 6⟶7 is a critical decision for the performance. As shown in Figure 8(a), among three candidate paths (colored blue), the shortest path (solid path) has the first hop of 6⟶7 while the others (dashed path) have 6⟶4. Thus, Metis discovers that selecting the first hop is important in deciding the path from 6 to 9, and 6⟶7 is selected. (ii) *Path a is less congested than path b.* For example, for connection #3 in Table 3, there are two paths with the same length, as shown in Figure 8(b). However, according to the traffic load, link 11⟶12 is severely congested. Therefore, path 7⟶10⟶11⟶12 should be avoided. Metis correctly identifies the critical branch and finds that 10⟶9 is an important decision to avoid the congested path (the red link in Figure 8(b)).

Besides the individual interpretations over connections, we also analyze the overall behaviors of Metis. We present the distribution of the mask values in Figure 9(a). Results demonstrate our optimization goal in §4.2 that the number of median mask values is reduced so that network operators can focus on the most critical connections. We also sum up all mask values on each link (vertex in the hypergraph) $\sum_e W_{v,e}$, and measure their relationship with the traffic on each link. As shown in Figure 9(b), 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 correlated, which indicates that the interpretations provided by Metis are reasonable. Note that Metis can provide connection-level interpretations as presented above, which is finer-grained than the information from link traffic.

6.2 Guide for Model Design

We present a use case to demonstrate that the interpretations of Metis can help the design of the DNN structure of Pensieve. As interpreted in §6.1, Metis finds that Pensieve significantly



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

Figure 12: For (a) and (b), Metis+Pensieve generates almost the same results with Pensieve, where 1200kbps and 2850kbps are rarely selected. (c) On a set of fixed-bandwidth links, 1200kbps and 2850kbps are still not preferred. Better viewed in color.

relies on the last chunk bitrate (r^t) when making decisions. This indicates that r^t may contain important information to the optimization of bitrate.

To utilize this observation, we modify the DNN structure of Pensieve to enlarge the influence of r^t on the output result. As shown in Figure 10(b), we directly concatenate the r^t to the output layer so that it can affect the prediction result more directly. Although the two DNN structures are mathematically equivalent, they will lead to different optimization performance and training efficiency due to the huge search space of DNNs [21]. After putting the significant feature nearer to the output layer (thus simplifying the relationship between the significant feature and results), the modified DNN will focus more on that significant feature.

We retrain the two DNN models on the same training and test sets and present the results in Figure 11. From the curves of the original model and the modified model, we can see that the modification in Figure 10 improves both the training speed and the final QoE. For example, on the test set, the modified DNN achieves 5.1% higher QoE on average than the original DNN. Considering the scale of views (millions of hours of video watched per day [68]) for video providers, even a small improvement in QoE is significant [42]. Moreover, the modified DNN can save 550k epochs on average to achieve the same QoE, which saves 23 hours on our testbed.

6.3 Enabling Debuggability

When interpreting Pensieve, we observe that some bitrates are rarely selected. The frequencies of selected bitrates of the experiments in §6.1 are presented in Figures 12(a) and 12(b). Among six bitrates from 300kbps to 4300kbps, two bitrates (1200kbps and 2850kbps) are rarely selected by Pensieve. The imbalance raises our interests since missing bitrates are *median* bitrates: the highest or lowest bitrates may not be selected due to network conditions, but not median ones.

To further explore the reasons, we emulate Pensieve on a set of links with fixed bandwidth ranging from 300kbps to 4500kbps. As the sample video used by [44] is too short for illustration, we replace the test video with a video of 1000 seconds and keep all other configurations the same with the original experiment. As shown in Figure 12(c), 1200kbps and 2850kbps are still not preferred by Pensieve. For example, on

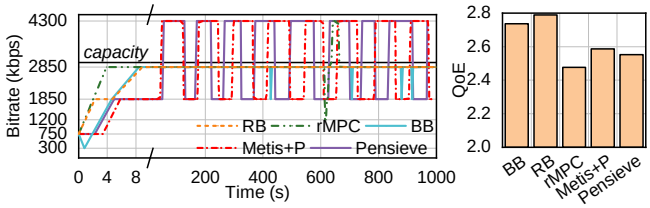


Figure 13: On a 3000kbps link, BB, RB, and rMPC learn the optimal policy and converge to 2850kbps. Metis+Pensieve (Metis+P) and Pensieve oscillate between 1850kbps and 4300kbps, degrading the QoE. Better viewed in color.

a fixed 3000kbps[4] link, the optimal decision of which should always select 2850kbps. However, in this case, only 0.4% of selections made by Pensieve are 2850kbps, while the remaining decisions are divided between 1850kbps and 4300kbps. As shown in Figure 13, Pensieve oscillates between 1850kbps and 4300kbps, which is also mimicked by Metis+Pensieve. However, such a policy is sub-optimal. In contrast, other baselines learn the optimal selection policy and fix their decisions to 2850kbps, achieving a higher QoE. Similar observations can also be observed on a 1200kbps link (Appendix D).

Studying the raw outputs of Pensieve, we find that Pensieve does not have enough confidence in either choice and therefore oscillates between them. The probability of selecting the optimal bitrate is at a surprisingly low level (Figure 25 in Appendix D). The training mechanism of Pensieve may cause this problem. At each step, the agent tries to *reinforce* particular actions that lead to larger rewards. 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 continuously selecting those actions and finally abandon the others. 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 Metis, since Pensieve is designed based on RL, network operators do not have an explicit dataset of bitrates. Network operators may have to penalize the imbalance of bitrate in the environment and retrain the DNN model for several hours to days. With Metis, the conversion from DNN to decision tree exposes an interface for network operators to debug the model. Since the dataset $\mathcal{D}$ to train the decision

[4]The goodput (bitrate) in this case is roughly 2850kbps.

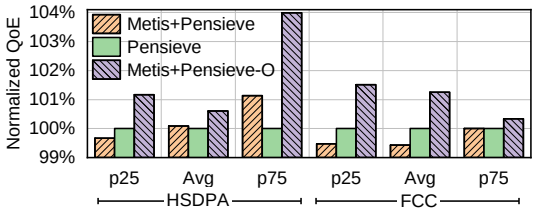
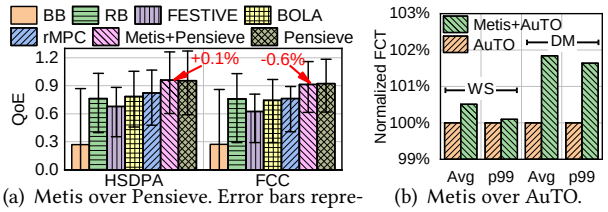


Figure 14: When converting DNNs to decision trees in Metis, oversampling the missing bitrates (Metis+Pensieve-O) improves the QoE by around 1% on average compared to the original DNN in Pensieve. QoE is normalized by Pensieve.



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

(b) Metis over AuTO.

Figure 15: The performance degradation between the original DNN and the decision tree interpreted by Metis is less than 2% for Pensieve and AuTO.

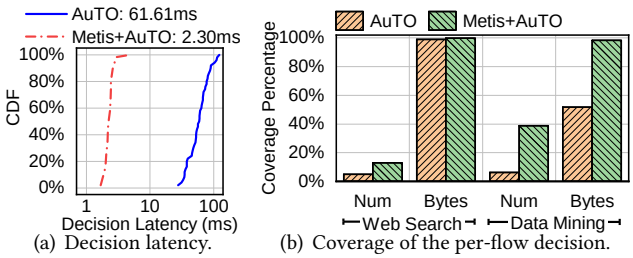
tree is highly *imbalanced*, as a straightforward solution, we oversample the missing bitrates to make sure their frequencies after sampling are around 1%. As shown in Figure 14, the oversampled decision tree (Metis+Pensieve-O) outperforms DNNs by about 1% on average and 4% at the 75^{th} percentile on HSDPA traces.

6.4 Lightweight Deployment

For local systems, decision trees provided by Metis are also lightweight to deploy. We first demonstrate that the performance degradation between the decision tree and the original DNN is negligible (less than 2%). Therefore, directly deploying decision trees of Pensieve and AuTO online will (i) shorten the decision latency, (ii) reduce the resource consumption and bring further performance benefits, and (iii) enable implementations onto advanced devices.

Performance maintenance. The performance of Metis-based systems is comparable to the original systems for both Pensieve and AuTO. As shown in Figure 15(a), the differences in average QoE between the decision tree interpreted by Metis and the original DNN of Pensieve are less than 0.6% on both traces. Similarly, as shown in Figure 15(b), the decision tree interpreted from AuTO (Metis+AuTO) degrades the performance within 2% compared to the original DNN. The performance loss is much less than the gain of introducing DNN (Pensieve by 14%, AuTO by up to 48%). Therefore, Metis could maintain the performance of the original DNNs with negligible degradation.

Decision latency. We showcase how Metis helps improve the decision latency of AuTO. The per-flow decision latency of AuTO is 62ms on average, during which short flows in data centers will run out. Converting DNNs into decision trees enables us to make per-flow decisions for more flows since



(a) Decision latency.

(b) Coverage of the per-flow decision.

Figure 16: By converting DNNs to decision trees, Metis could (a) shorten the decision latency by 26.8×, and therefore (b) enlarge the coverage of the per-flow decision.

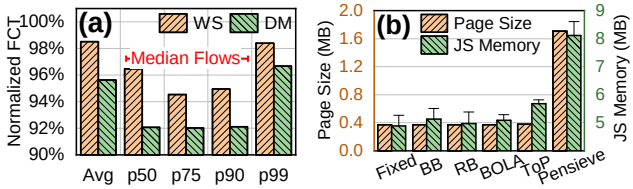


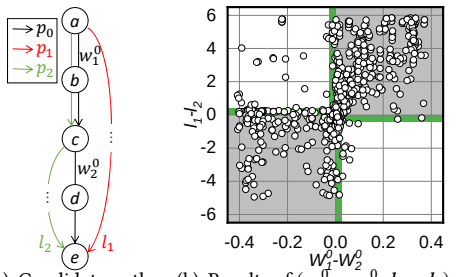
Figure 17: (a) With precise per-flow optimizations, Metis+AuTO could reduce the FCT for median flows. FCT is normalized by AuTO. (b) Compared to the original Pensieve model, Metis+Pensieve could reduce both page size and JS memory.

the decision latency is shortened. As shown in Figure 16(a), when replacing DNNs with decision trees, the decision latency of per-flow scheduling could be reduced by 26.8×. In this case, compared to AuTO, Metis+AuTO will cover more flows by 33% and more bytes by 46% for DM traces [14], as shown in Figure 16(b).

By covering more flows, Metis+AuTO can perform optimized per-flow scheduling for not only long flows but also *median flows*, which will improve the overall performance. We modify our prototype of Metis+AuTO to allow the decision tree to schedule median flows and present the FCT results in Figure 17(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% on two traces. We also observe significant performance improvements for median flows (from the 50^{th} to the 90^{th} percentile) by up to 8.0%. This indicates that median flows enjoy the benefits of precise per-flow scheduling. Improvements in DM traces are better than WS since the coverage increase of DM is larger than that of WS (cf. Figure 16(b)).

Resource consumption. We then evaluate the resource consumption (specifically, *page load time* and *memory consumption*) of Metis+Pensieve. As there are other modules in the player, we compare ABR algorithms with the `fixed` algorithm, which always selects the lowest bitrate.

For 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 17(b), Fixed, BB, RB, and BOLA have almost the same page size because of their simple processing logic. Pensieve increases the page size by 1370KB since it needs to download the DNN model first. In contrast, Metis+Pensieve has a similar page size with the heuristics.



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

Figure 18: For RouteNet, the mask value provided by Metis (e.g., $w_1^0 > w_2^0$) could help network operators select the better path (l_2) during ad-hoc adjustments.

When the goodput is 1200kbps (the average bandwidth of Pensieve's evaluation traces), the *additional* page load time of ABR algorithms compared to `fixed` is reduced by 156×: Pensieve introduces an additional page load time of 9.36 seconds, while Metis+Penseve only adds 60ms.

We then measure the runtime memory and present the results in Figure 17(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 Metis+Pensieve is reduced by 4.0× on average and 6.6× on the peak, which is at the same level as other heuristics.

On-device implementation. Besides, converting DNNs into decision trees also make the model implementable on data plane devices. For example, DNNs are hardly possible to be implemented even with advanced devices (e.g., SmartNICs [53] and programmable switches [10]) since there are a lot of complicated operations (e.g., floating numbers) [11]. In contrast, decision trees could be implemented with branching clauses only. This enables the offloading of decision trees onto data planes devices. We preliminarily demonstrate the potential by implementing the decision tree onto a Netronome NFP-4000 SmartNIC [53]. The decision tree interpretations enable us to deploy the Metis+AuTO-lRLA with 1,000 LoCs. Evaluation results also show that the decision latency of Metis+AuTO on SmartNICs 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 [10] and the comparison with other baselines for future work.

6.5 Ad-Hoc Adjustments

We present a use case of Metis on how network operators can execute ad-hoc adjustments onto RouteNet* based on the interpretations of Metis. In the routing case, network operators might need to reroute a flow to another path due to external reasons (e.g., pricing). As shown in Figure 18(a), when the demand from node a to node e needs rerouting away from the original path p_0, there are several candidates paths (p_1 and p_2). Since network operators do not know the actual performance of each path until rerouting rules are installed, deciding which path to reroute is challenging. This scenario with multiple equal-cost paths is common in topologies such as fat-trees.

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 diverting node and its next-hop link. For example, in Figure 18(a), p_1 diverts from p_0 at node a and p_2 diverts from p_0 at node c. w_1^0 is the mask value of the connection between p_0 and link $a \rightarrow b$. Since w_1^0 represents the significance of selecting $a \rightarrow b$ rather than other links, it is correlated to the possibility that there is also a relatively good path if $a \rightarrow b$ is not selected. Recalling the optimization in Equation 4, a lower mask value of a connection means that the selection is not critical 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 8. Since the optimal path may not be the shortest path, we consider all paths that are $\leqslant$1 hop longer than the shortest path as candidates. For example, for path 0→2→5→12, path 0→3→4→5→12 is considered as a candidate, but path 0→1→ 7→10→11→12 is not. We go through all pairs of demand in the NSFNet topology in Figure 8 and measure all the path latency for such candidate scenarios and the mask values at the diverting node. We repeat the experiments with all the 50 traffic samples provided by [59].

For each routing path p_0 generated by RouteNet*, we collect all (p_0, p_1, p_2) that satisfy the conditions in Figure 18(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 18(b) (750 points in total). Most points (72%) fall into quadrants I and III (shaded gray) with another 19% points very close to quadrants I and III (shaded green), which verifies our observation above. Thus, we provide an indicator for network operators to decide which path to reroute without estimating end-to-end path latency.

6.6 Metis Deep Dive

Finally, we overview the experiments that benchmark the interpretability of Metis. The detailed experimentation setup and more empirical results are deferred to the appendix.

Interpretation baselines comparison. We compare the performance of the decision tree in Metis against two baselines in the DL community. We implement LIME [55], one of the most typical blackbox interpretation methods in the DL community, and LEMNA [27], an interpretation method specifically designed for time sequences in RNN. We measure the misprediction rate and errors of three interpretation methods. The misprediction rates on two systems with Metis-based methods are reduced by 1.2×-1.7× compared to two baselines. The root-mean-square errors (RMSEs) are reduced by 1.2×-3.2×. Experiments are presented in Appendix E in

detail. The decision tree outperforms the other two interpretation methods, which confirms our design choice in §3.1.

Sensitivity analysis. We test the robustness of hyperparameters of Metis in Appendix F. For decision tree interpretations, we test the robustness of the number of leaf nodes. Results show that a wide range of settings (from 10 to 5000) perform well for Pensieve and AuTO (accuracy variations within 10%). For hypergraph interpretations, we vary the two hyperparameters λ_1 and λ_2 in Equation 4. We then measure how the interpreted mask values respond to the variations of hyperparameters. Results show that network operators could effectively adjust respective hyperparameters according to their needs. For example, when network operators want to inspect less critical connections, they can increase the value of λ_1 to penalize the scale of mask values.

Computation overhead. In Appendix G, our evaluation shows that converting finetuned DNNs into decision trees for Pensieve and AuTO takes less than 40 seconds under different settings. For hypergraph interpretations, the computation time of generating the mask values for RouteNet* is 80 seconds on average. This offline computation time is negligible compared to the training time of DNN models, which may take several hours to days.

7 Discussion

In this section, we discuss the limitations and potential future directions of Metis.

Can Metis interpret *all* networking systems? Admittedly, Metis cannot interpret all DL-based networking systems. For example, network intrusion detection systems (NIDSes) are used to detect malicious packets with regular expression matching on the packet payload [50]. Prior DL-based methods introduced RNN to improve the performance of NIDSes [72]. However, since RNN (and other DNNs with recurrent structures) fundamentally contains *implicit* memory units, decision trees cannot faithfully capture the policy with only *explicit* decision variables. In the future, we aim to combine Metis with recurrent units, e.g., employing recurrent decision trees [13]. We also clarify the capability of hypergraph formulation in §4.2.

Why not directly train a simpler model? As shown in §6.4, converted simpler models (e.g., decision trees) exhibit comparable performance to larger models. However, *directly* training the simpler model from scratch is difficult to achieve the same performance [44]. One possible explanation behind this phenomenon is the *lottery ticket hypothesis* [23, 75]: training deep models is analogous to winning the lottery by buying a very large number of tickets (i.e., building a large neural network). However, we cannot know the winning ticket configuration in advance. Therefore, directly training a simpler model is similar to buying one lottery ticket only, which has little chance to achieve satisfying performance.

Will interpretations always be correct? Metis is designed to offer a sense of confidence by helping network operators understand (and further troubleshoot) DL-based networking systems. However, the interpretations themselves can also make mistakes. In fact, researchers have recently discovered attacks against the interpreting systems for image classification [28, 81]. Nonetheless, interpretations from our experiments are empirically sane (§6). Since the interpretations are concise and well understood, human operators could easily spot the rare case of erroneous interpretation.

8 Related Work

There is little prior work on interpreting DL-based networking systems. Some position papers discuss the problems and preliminary solutions for interpreting DL-based networking systems [18, 82]. In terms of approach, the closest work is NeuroCuts [40], which optimizes a decision tree for packet classification with deep RL and is therefore self-interpretable. However, NeuroCuts directly trains the decision tree from scratch for packet classification only while Metis interprets existing diverse DL-based networking systems. In the following, we survey the practicality of prior work on using DL in networking applications and alternative methods to apply DNNs interpretations in other domains.

Practicality of DL-based networking systems. There are also some other issues of DL-based networking systems that need to be addressed before deployed in practice. Some recent work focuses on the verification of DL-based networking systems [34], which is orthogonal to our work and could be adopted together for a more practical system. Recent solutions also address the heavyweight issue of specific networking systems [8, 30, 46], which do not focus on interpretability and are difficult to support complex DL-based networking systems. Metis provides a systematic solution to effectively interpret diverse DL-based networking systems with high quality for practical deployment. Metis could also be integrated with research efforts on the training phase of DL-based networking systems [43] to achieve a practical system at the design phase.

Interpretation methods. As discussed in §2.2, many interpretability approaches focus on understanding the mechanism of DNNs, such as convolutional neural networks (CNN) [7], RNN [27], GNN [74], which is not the goal of Metis. Besides interpretation methods introduced in §2.2 and §3.1, there are also some research efforts to interpret existing applications in many domains. Examples include image analysis [7, 76], neural language translation [56, 65], recommendation systems [12, 15], and security applications [27, 81]. However, as discussed in §2.2, existing methods are insufficient for networking systems. There still lacks an effective interpretation method for the networking community. Metis sheds light on the interpretability of DL-based networking systems with our specially designed framework.

Hypergraph learning. In the machine learning community, the hypergraph structure has many applications in the modeling of high-order correlation in social network and image recognition. The message passing process on hypergraph

structure is first introduced in [83]. The most recent efforts combine the hypergraph structure with convolution [22, 70] and attention mechanisms [80] to further improve the model performance. The objective there is to *directly optimize* the model performance (e.g., prediction accuracy). In contrast, Metis employs the hypergraph structure to *formulate* existing networking system outputs and *interpret* the critical components in hypergraphs. A possible direction is to *design* more DL-based networking systems with our hypergraph formulation (§4.1) and hypergraph learning methods above, which is left as our future work.

9 Conclusion

In this paper, we propose Metis, a new framework to interpret diverse DL-based networking systems. Metis categorizes DL-based networking systems and provides respective solutions by modeling and analyzing the commonplaces of them. We apply Metis over several typical DL-based networking systems. Evaluation results show that Metis-based systems can interpret the behaviors of DL-based networking systems with high quality. Further use cases demonstrate that Metis could help network operators design, debug, deploy, and ad-hoc adjust DL-based networking systems.

This work does not raise any ethical issues.

References

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

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

[3] 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.

[4] 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.

[5] 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.

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

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

[8] Daniel S Berger. Towards lightweight and robust machine learning for cdn caching. In *Proc. ACM HotNets*, 2018.

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

[10] 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.

[11] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests. *arXiv preprint 1909.05680*, 2019.

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

[13] Jianhui Chen, Hoang M Le, Peter Carr, Yisong Yue, and James J Little. Learning online smooth predictors for realtime camera planning using recurrent decision trees. In *Proc. IEEE CVPR*, 2016.

[14] 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.

[15] Xu Chen, Yongfeng Zhang, and Zheng Qin. Dynamic explainable recommendation based on neural attentive models. In *Proc. AAAI*, 2019.

[16] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshop on Deep Learning*, 2014.

[17] 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.

[18] 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.

[19] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Commun. ACM*, pages 68–77, 2020.

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

[21] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 2019.

[22] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proc. AAAI*, 2019.

[23] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proc. ICLR*, 2019.

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

[25] 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.

[26] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)*, 2018.

[27] 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.

[28] Juyeon Heo, Sunghwan Joo, and Taesup Moon. Fooling neural network interpretations via adversarial model manipulation. In *Proc. NeurIPS*, 2019.

[29] 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.

[30] 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.

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

[32] 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.

[33] Mahmoud Kamel, Walaa Hamouda, and Amr Youssef. Ultra-dense networks: A survey. *IEEE Communications Surveys & Tutorials*, 2016.

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

[35] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proc. ICLR*, 2014.

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

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

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[39] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[40] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. Neural packet classification. In *Proc. ACM SIGCOMM*, 2019.

[41] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of Berkeley symposium on mathematical statistics and probability*, 1967.

[42] Hongzi Mao, Shannon Chen, Drew Dimmery, Shaun Singh, Drew Blaisdell, Yuandong Tian, Mohammad Alizadeh, and Eytan Bakshy. Real-world video adaptation with reinforcement learning. In *ICML Reinforcement Learning for Real Life Workshop*, 2019.

[43] 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.

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

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

[46] Zili Meng, Jing Chen, Yaning Guo, Chen Sun, Hongxin Hu, and Mingwei Xu. Pitree: Practical implementation of abr algorithms using decision trees. In *Proc. ACM MM*, 2019.

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

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

[49] 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.

[50] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Elsevier Journal of network and computer applications*, pages 42–57, 2013.

[51] Shar Narasimhan. Nvidia clocks worldâĂŹs fastest bert training time and largest transformer based model. https://devblogs.nvidia.com/training-bert-with-gpus/, 2019.

[52] Srihari Nelakuditi, Sanghwan Lee, Yinzhe Yu, Zhi-Li Zhang, and Chen-Nee Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM Transactions on Networking*, pages 359–372, 2007.

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

[54] 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*, 2011.

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

[56] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In *Proc. ACL*, 2018.

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

[58] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proc. AISTATS*, 2011.

[59] 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.

[60] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 1948.

[61] 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.

[62] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. From theory to practice: improving bitrate adaptation in the dash reference player. In *Proc. ACM MMSys*, 2018.

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

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

[65] Mariya Toneva and Leila Wehbe. Interpreting and improving natural-language processing (in machines) with natural language-processing (in the brain). In *Proc. NeurIPS*, 2019.

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

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

[68] Kurt Wagner. Facebook says video is huge – 100-million-hours-per-day huge. https://www.vox.com/2016/1/27/11589140/, 2016.

[69] 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.

[70] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. In *Proc. NeurIPS*, 2019.

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

[72] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, pages 21954–21961, 2017.

[73] 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.

[74] 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.

[75] Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *Proc. ICLR*, 2020.

[76] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proc. ECCV*, 2014.

[77] Chun Zhang, Yong Liu, Weibo Gong, Jim Kurose, Robert Moll, and Don Towsley. On optimal routing with multiple traffic matrices. In *Proc. IEEE INFOCOM*, 2005.

[78] Hongliang Zhang, Lingyang Song, Yonghui Li, and Geoffrey Ye Li. Hypergraph theory: Applications in 5g heterogeneous ultra-dense networks. *IEEE Communications Magazine*, 55(12):70–76, 2017.

[79] Menghao Zhang, Jiasong Bai, Guanyu Li, Zili Meng, Hongda Li, Hongxin Hu, and Mingwei Xu. When nfv meets ann: Rethinking elastic scaling for ann-based nfs. In *Proc. IEEE ICNP*, 2019.

[80] Ruochi Zhang, Yuesong Zou, and Jian Ma. Hyper-sagnn: a self-attention based graph neural network for hypergraphs. In *Proc. ICLR*, 2020.

[81] Xinyang Zhang, Ningfei Wang, Shouling Ji, Hua Shen, and Ting Wang. Interpretable deep learning under fire. In *Proc. USENIX Security*, 2020.

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

[83] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Proc. NIPS*, 2007.

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

Appendices

A Resampling in Decision Tree Training

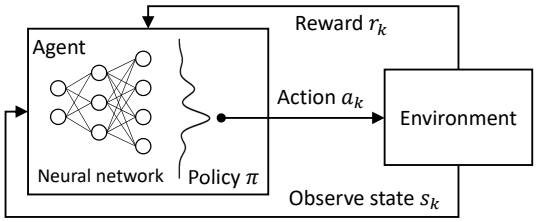


Figure 19: RL with neural networks as policy.

To explain the resampling equation during decision tree training (Equation 1), we first briefly introduce the basic knowledge about RL used in this paper. We refer the readers to [64] for a more comprehensive understanding of RL.

In RL, at each iteration t, the *agent* (e.g., a flow scheduler [14]) first observers a *state* $s_t \in \mathcal{S}$ (e.g., remaining flow sizes) from the surrounding *environment*. The agent then takes an *action* $a_t \in \mathcal{A}$ (e.g., scheduling a flow to a certain port) according to its *policy* π (e.g., shortest flow first). The environment then returns a *reward* r_t (e.g., FCTs of finished flows) 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* $\mathbb{E}\left[\sum_t \gamma^t r_t\right]$ 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 [48, 49]. An illustration of RL is presented in Figure 19.

However, it is not easy for the agent to find out the actual reward of a state or an action in the training process since the reward is usually *delayed*. For example, the FCT can only be observed after the flow is completed. Therefore, we need to estimate the potential *value* of a state. *Value function* $V_t^{(\pi)}(s)$ is introduced to determine the potential future reward of a state s at the time t with the policy π:

$$V^{(\pi)}(s) = R(s) + \sum_{s' \in \mathcal{S}} p\left(s'|s, \pi(s)\right) V^{(\pi)}(s') \tag{10}$$

where $p(s'|s, a)$ is the transition probability onto state s' given state s and subsequent action a. Similarly, *Q-function* $Q_t^{(\pi)}(s, a)$ is to estimate the value of how a certain action a at state s may contribute to the future reward:

$$Q^{(\pi)}(s, a) = R(s) + \sum_{s' \in \mathcal{S}} p\left(s'|s, a\right) V^{(\pi)}(s') \tag{11}$$

Therefore, a good action a at the state s would maximize the difference between the value function and Q-function, i.e., the optimization loss $\ell(s, \pi)$ of RL could be written as:

$$\ell(s, \pi) = V^{(\pi)}(s) - Q^{(\pi)}(s, a) \tag{12}$$

In the teacher-student learning optimization in §3.2, to make the loss independent of π and therefore easy to optimize, Bastani et al. [6] bounded the loss above with:

$$\tilde{\ell}(s) = V^{(\pi)}(s) - \min_{a' \in A} Q^{(\pi)}(s, a') \geqslant V^{(\pi)}(s) - Q^{(\pi)}(s, a) \tag{13}$$

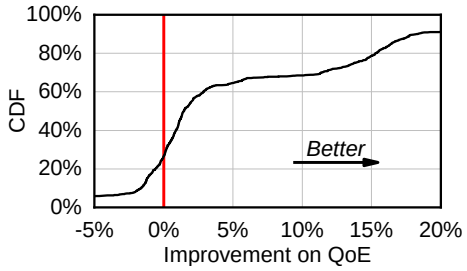
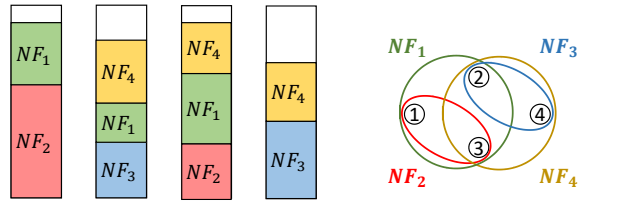


Figure 20: The resampling step could improve the QoE of 73% of the traces, with the median improvement of 1.5%.



(a) Original representation. (b) Hypergraph representation.

Figure 21: Network function virtualization.

Therefore, we can resample the (state, action) pairs with the loss function above, which explains the sampling probability in Equation 1. The sampling probability $p(s, a)$ in Equation 1 is proportional to but not equal to the loss function due to the normalization of probability.

We further empirically evaluate the improvement on QoE of the resampling step. We measure the QoE of the decision trees with and without the resampling step. As shown in Figure 20, 73% of traces could benefit from the resampling step with different degrees of improvement. The median improvement on QoE over all traces is 1.5%. Since the resampling step is adopted for the last mile performance improvement, network operators may choose to skip the step if performance is not a critical issue for them.

B Hypergraph Formulations

We present several other formulations of different application scenarios presented in Table 2 (§4.1).

B.1 Network Function Placement System

Network function virtualization (NFV) is widely adopted to replace dedicated hardware with virtualized network functions (VNFs). Considering the processing ability and fluctuating network demand, network operators can replicate one VNF onto several instances on different servers, and consolidate multiple VNFs onto one server [69]. 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. An illustration of the hypergraph formulation of NFV placement is presented in Figure 21. Hyperedge e contains with vertex v indicates that VNF v has an instance placed onto server e. Hyperedge features F_E could be the

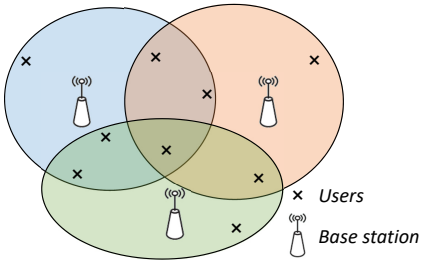


Figure 22: Ultra-dense network.

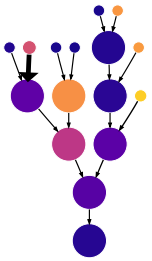


Figure 23: Cluster scheduling jobs.

processing capacity of servers, and vertex features F_V could be the processing speed of different types of VNF. Metis will then interpret the placement results by finding the critical NF instance placement and checking if it is reasonable.

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 networking* [33]. 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 formulated as a hypergraph [78], with the coverage of the picocell base stations as hyperedges, and mobile users as vertices.

We present an illustration of the hypergraph formulation in Figure 22. 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. Metis could consequently interpret the system by providing insights on which user-base station connection is critical to the overall performance. For example, connecting a user with high demands to a nearby base station indicates that the system might mimic a nearest-first policy.

B.3 Cluster Scheduling System

In cluster scheduling scenarios (e.g., Spark [45]), a job consists of a directed acyclic graph (DAG) whose nodes are the execution stages of the job, as shown in Figure 23. 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. Since the jobs to schedule are usually represented as dataflow graphs [45], we can naturally formulate the cluster scheduling scenario with Metis. In this case, each vertex represents a set of parallel operations, and each edge represents the dependency between vertices. Vertex features F_V are the work of nodes, and hyperedge features F_E are the data transmission between nodes [45]. Metis can interpret the scheduling system by finding out which scheduling decisions (allocating a specific job node to a certain number of executors) are significant to the performance.

Pensieve	M	200
AuTO	M (lRLA)	2000
	M (sRLA)	2000
RouteNet*	λ_1	0.25
	λ_2	1

Table 4: Metis hyperparameters.

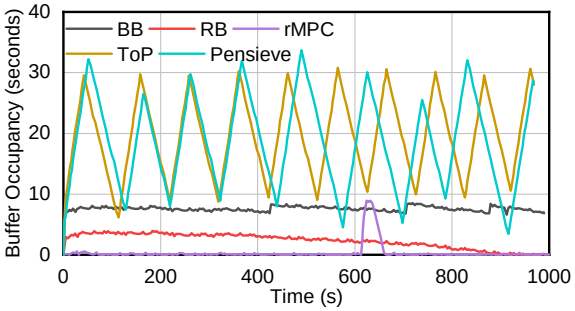


Figure 24: Buffer Occupancy at 3000kbps Link.

C Implementation Details

We introduce the parameter settings of Metis and three DL-based networking systems, together with the details of our testbed, in this section.

Parameter settings. We present the hyperparameter settings of Metis in Table 4. For the DNN in Pensieve, we set the number of leaf nodes (M) to 200. Our experiments on the sensitivity of M in Appendix F.1 shows that a wide range of M perform well. 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 Pensieve (25 states).

For the hypergraph-based interpretation method, network operators can set the hyperparameters λ_1 and λ_2 based on their ability to understand the interpreted structure and application scenarios, as discussed in §4.2. For example, with the settings in Table 4 results, only 10% of connections of RouteNet* have mask values greater than 0.8. Further improving λ_1 will increase the ratio of connections with high mask values and expose more critical connections to network operators. We present the details of sensitivity analysis of λ_1 and λ_2 in Appendix F.2.

Note that the five hyperparameters in Table 4 are the hyperparameters for three systems in total. In practice, network operators only need to set one or two to employ Metis on their own DL-based networking system.

Testbed details. We train the decision tree with `sklearn` [54] and modify it to support the CCP. The server for Pensieve and RouteNet* is equipped with an Intel Core i7-8700 CPU (6 physical cores) and an Nvidia Titan Xp GPU. The switchs used in AuTO are two H3C-S6300 48-port switches.

D Pensieve Debugging Deep Dive

We also provide more details on the experiments of two links with bandwidth fixed to 3000kbps and 1300kbps in §6.3.

3000kbps link. Except for the experiments in §6.3, we also investigate the runtime buffer occupancy over the 3000kbps

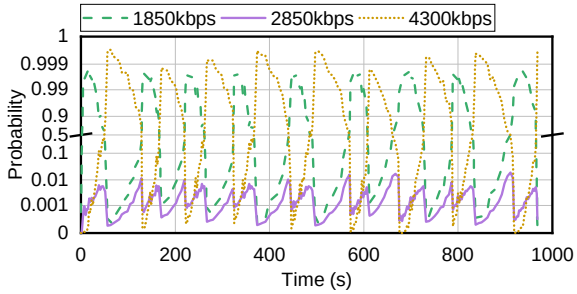


Figure 25: Probabilities of selecting 1850kbps, 2850kbps, 4300kbps qualities. The probability of selecting other three qualities is less than 10^{-4} thus not presented.

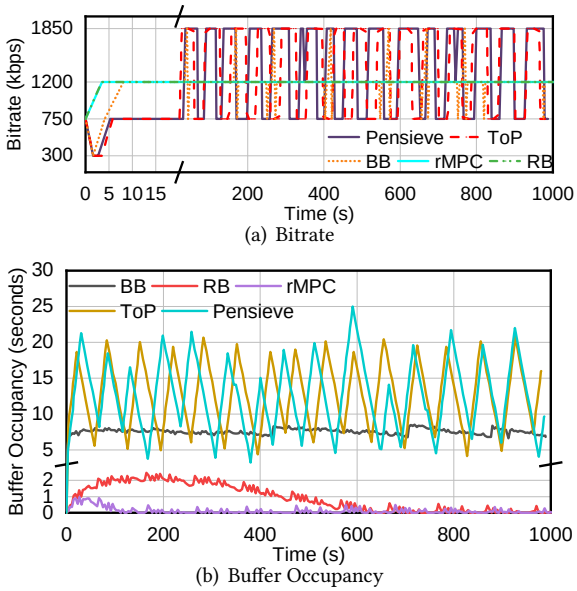


Figure 26: Results on a 1300kbps link. Better viewed in color.

link. As shown in Figure 24, the buffer occupancy of Pensieve fluctuates: buffer increases when 1850kbps is selected and decreases when 4300kbps is selected, which is also faithfully mimicked by Metis+Pensieve. The oscillation leads to a drastic smoothness penalty. Meanwhile, the buffer occupancy can also interpret the poor performance of rMPC in Figure 13: 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 the same. Thus a substantial 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 normalized probabilities of selecting each action, we further investigate those probabilities of Pensieve on the 3000kbps link and present the results in Figure 25. A higher probability close to 1 indicates higher confidence in the decision. We can see that Pensieve does not have enough confidence in the decision it made, which suggests that Pensieve might not experience similar conditions in training; thus, it does not know how to make a decision.

BB	RB	rMPC	Metis+Pensieve	Pensieve
1.050	0.904	0.803	0.986	0.983

Table 5: QoE on the 1300kbps link.

1300kbps link. We also provide the details about the experiments in Figure 12(c) on a 1300kbps link and present the results in Figure 26 and Table 5. The results are similar to the 3000kbps experiment, except that the performance of RB is worse since it converges faster.

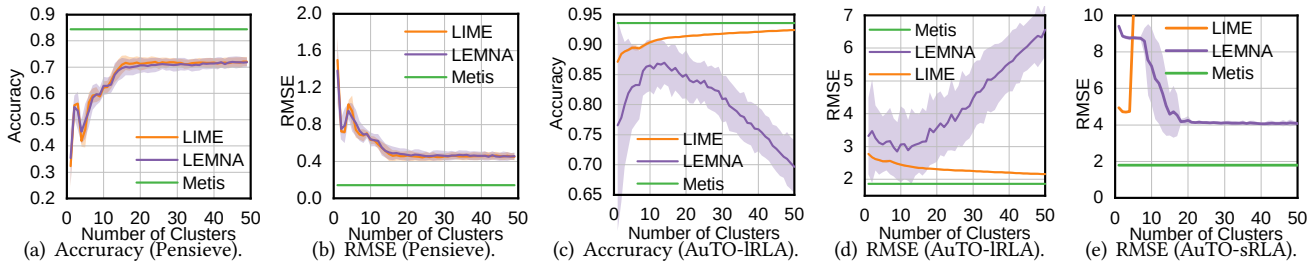
E Interpretation Baseline Comparison

We further want to know the reason for the performance maintenance of Metis. We measure the accuracy and root-mean-square error (RMSE) of the decisions made by Metis compared to the original decisions made by DNNs. As baselines, we compare the faithfulness of Metis over three DNNs (Metis+Pensieve, Metis+AuTO-lRLA, Metis+AuTO-sRLA) with two recent interpretation methods:

- LIME [55] is one of the most widely used blackbox interpretation method in the machine learning community. LIME interprets the blackbox model with the linear regression of the inputs and outputs.
- LEMNA [27] is an interpretation method proposed in 2018 and designed to interpret DL models based on time-series inputs (e.g., RNN). LEMNA employs a mixture regression to handle the dependency between inputs. We employ LEMNA as a baseline since some networking systems also handle time-series inputs.

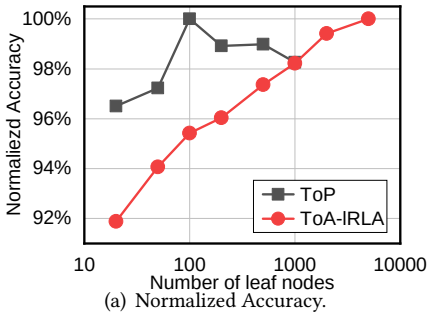
As both methods are designed based on regressions around a certain sample, to make a fair comparison, we run the baselines in the following way: At the training stage, we first use k-means clustering [41] to cluster the input-output samples of the DL-based networking system into k groups. We then interpret the results inside each group with LIME and LEMNA. 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 27. Since the decision tree interpretations of Metis do not rely on a particular sample, they do not need to be clustered and are constant lines.

From Figures 27(a) and 27(c), Metis+Pensieve and Metis+AuTO-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 [4, 5] are much simpler than those of video bitrate adaption (e.g., stochastic optimization [73], Lyapunov optimization [63]), the accuracy of Metis+AuTO-lRLA is a little higher than that of Metis+Pensieve. The low decision errors in Figures 27(b), 27(d), and 27(e) indicate that even for those decision tree decisions that are different from DNNs, the error made by Metis 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 §6.4. Meanwhile, the accuracy and RMSE of Metis are much better than those of LIME and LEMNA. Our design choice in §3.1 is thus verified: decision trees can

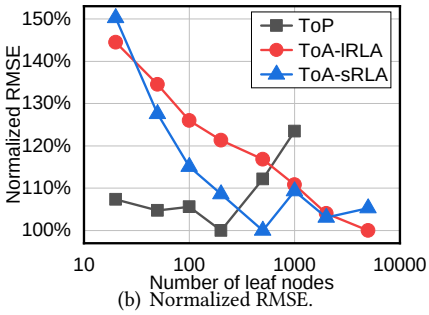


(a) Accuracy (Pensieve). (b) RMSE (Pensieve). (c) Accuracy (AuTO-lRLA). (d) RMSE (AuTO-lRLA). (e) RMSE (AuTO-sRLA).

Figure 27: Faithfulness of Metis. Shaded area spans ± std. AuTO-sRLA predicts real values thus does not have accuracy. Results of LIME over sRLA diverges with ⩾5 clusters. Higher accuracy and lower RMSE indicate a better performance. Better viewed in color.

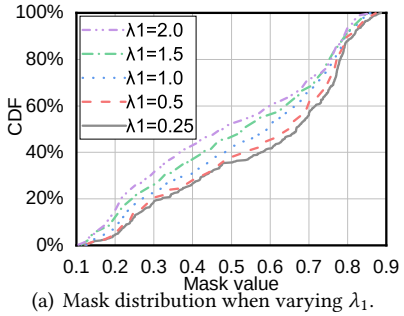


(a) Normalized Accuracy.

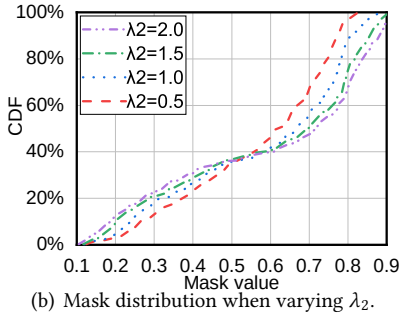


(b) Normalized RMSE.

Figure 28: Sensitivity of leaf nodes on prediction accuracy and RMSE. Results are normalized by the best value on each curve.



(a) Mask distribution when varying λ_1.



(b) Mask distribution when varying λ_2.

Figure 29: The masks optimized by Metis could effectively respond to the variation of hyperparameters λ_1 and λ_2.

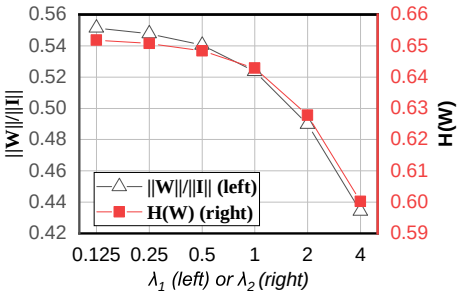


Figure 30: The value of different terms in Equation 4 reacts to the change of λ_1 and λ_2.

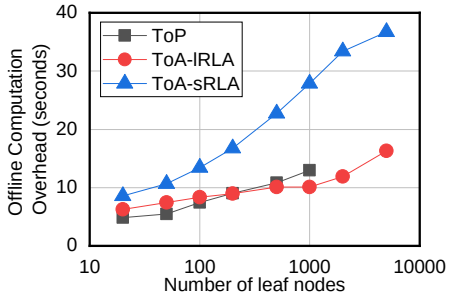


Figure 31: Offline Computation Overhead of Metis with different number of leaf nodes.

provide richer expressiveness and are suitable for networking systems. The performance of LEMNA is unstable for two agents of AuTO since the states of AuTO is highly centralized at several places from our experiments, which degrades the performance of expectation-maximization iterations in LEMNA [27].

F Sensitivity Analysis

In this section, we present the sensitivity analysis results on the hyperparameters of Metis when applied to three DL-based networking systems.

F.1 Pensieve and AuTO

To test the robustness of Metis 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 E (Pensieve, AuTO-sRLA, AuTO-lRLA). The results are presented in Figure 28. The accuracy and RMSE of Metis+Pensieve with the number of leaf nodes varying from 20 to 5000 are better than the best results of LIME and LEMNA in Figure 27 in Appendix E. Metis+AuTO-lRLA and Metis+AuTO-sRLA outperform the best value of LIME and LEMNA in a wide range from 200 to 5000 leaf nodes. The robustness 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.

F.2 RouteNet*

We measure the sensitivity of two hyperparameters λ_1 and λ_2 when interpreting the hypergraph-based global systems with Metis as introduced in §4.2. As presented in Figure 29(a), when network operators increase λ_1, $||W||$ will therefore be penalized. The generated mask values will also be reduced, shifting the cumulative distribution curve up. Those essentially critical connections will be revealed to network operators. Experiments of varying λ_2 demonstrate similar results, as presented in Figure 29(b). A higher λ_2 will make mask values concentrated at 0 or 1, resulting in a steeper cumulative distribution curve.

We further measure how will the specific values in Equation 5 change when network operators vary the hyperparameters. We measure the $\frac{||W||}{||I||}$ (scale of 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 30. 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 30. From the results, we can see that different terms in the optimization goal all actively respond to the changes in respective hyperparameters.

G Computation Overhead

We further examine the computation overhead of Metis in decision tree extraction. We measure the decision tree computation time of Pensieve, AuTO-lRLA, and AuTO-sRLA at different numbers of leaf nodes on our testbed. As the action space of Pensieve (6 actions) is much smaller than those of AuTO-lRLA (108 actions) and AuTO-sRLA (real values), the decision tree of Metis+Pensieve has completely been separated with around 1000 leaf nodes. Thus we cannot generate decision trees for Metis+Pensieve with more leaf nodes without enlarging the training set. As shown in Figure 31, 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 [44] and 8 hours in AuTO [14]). Metis can convert the DNNs into decision trees with negligible computation overhead.

For RouteNet*, we also measure the computation time of the optimization of the mask matrix W. For hypergraph interpretations with 50 different traffic demand samples, the computation time of mask matrices is 80 seconds on average. For offline interpretations of RouteNet*, the computation time is negligible compared to the training time of DNNs. Even for online inspections, the interpretation time is acceptable for most cases since the routing information in a well-configured network rarely changes every tens of seconds [52]. In the future, we will also investigate the incremental computation of the mask values to further accelerate the interpretation.