End-to-End Performance Analysis of Learning-enabled Systems

Pooria Namyar[†], Michael Schapira[‡], Ramesh Govindan[†], Santiago Segarra[⋆],
Ryan Beckett[§], Siva Kesava Reddy Kakarla[§], Behnaz Arzani[§]
[†]University of Southern California, [‡]Hebrew University of Jerusalem, [⋆]Rice University, [§]Microsoft

Abstract

We propose a performance analysis tool for learning-enabled systems that allows operators to uncover potential performance issues *before* deploying DNNs in their systems. The tools that exist for this purpose require operators to faithfully model all components (a white-box approach) or do inefficient black-box local search. We propose a gray-box alternative, which eliminates the need to precisely model all the system's components. Our approach is faster and finds substantially worse scenarios compared to prior work. We show that a state-of-the-art learning-enabled traffic engineering pipeline can underperform the optimal by 6× — a much higher number compared to what the authors found.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Networks** → **Network performance analysis**; **Network reliability**; **Network management**.

Keywords

Machine Learning for Systems, Performance Analysis

ACM Reference Format:
Pooria Namyar, Michael Schapira, Ramesh Govindan, Santiago Segarra, Ryan Beckett, Siva Kesava Reddy Kakarla, Behnaz Arzani. 2024. End-to-End Performance Analysis of Learning-enabled Systems. In *The 23rd ACM Workshop on Hot Topics in Networks (HOTNETS '24), November 18–19, 2024, Irvine, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3696348.3696875

1 Introduction

Operators have started to use deep neural networks (DNNs) in the *computation pipeline* of many cloud systems [3, 24, 26, 30, 36, 44, 46, 47]. In these *learning-enabled systems*, DNNs run faster than optimal algorithms [36, 46] and perform better than heuristics when the optimal solution is either computationally expensive or unknown [26, 47].

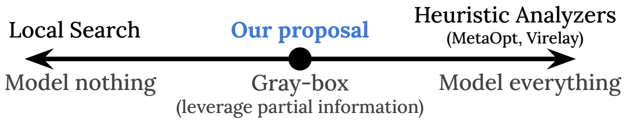


Figure 1: We propose a gray-box approach that uses gradient to analyze performance of learning-enabled systems. Existing white-box heuristic analyzers have limited expressiveness and scalability [13, 32, 33] as they require an exact model of the system (including the DNN) in optimization or logic. Local search methods fail to find bad inputs as they ignore all the details about the underlying system.

Operators train and test individual DNNs for common workloads and network conditions. However, they cannot check whether the *end-to-end learning-enabled system* (which includes the DNN and other computational components) performs well across all possible inputs. They need tools that can *quickly* answer questions such as: How much can the performance of a learning-enabled system deviate from the optimal or other baselines? What inputs cause learning-enabled systems to underperform? Are there in-distribution inputs (those from the same distribution as the training set) that would cause the DNN-based system to underperform?

It is hard to build such a tool (§3). Verification techniques (DNN verifiers) ensure DNN's correctness by proving specific invariants on the DNN in isolation [10, 22, 43] rather than evaluating the performance of the entire system. *White-box* heuristic analyzers [13, 32, 33] can analyze the end-to-end performance in theory but require an *exact* model of the DNN and other components of the system, either in optimization or in first-order logic. This requirement limits their expressiveness and scalability. For instance, we find that MetaOpt [33] is unable to analyze a state-of-the-art learning-enabled traffic engineering system (see §5).

Local *black-box* search algorithms apply to any system, including learning-enabled ones. However, they cannot efficiently search large input spaces and often fail to find practical adversarial inputs [32, 33]. These methods neglect all the

HOTNETS '24, November 18–19, 2024, Irvine, CA, USA

ACM ISBN 979-8-4007-1272-2/24/11
https://doi.org/10.1145/3696348.3696875

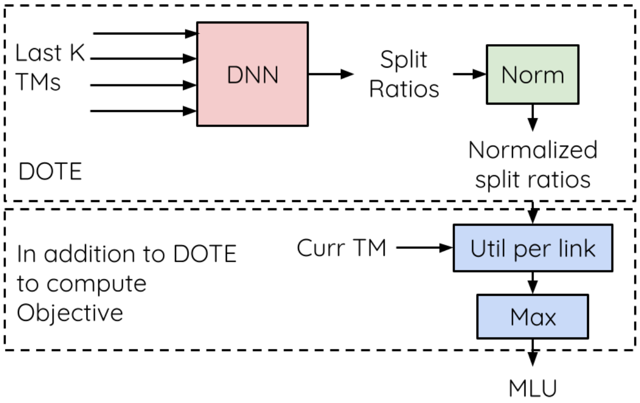


Figure 2: A Traffic Engineering pipeline that uses DOTE to route the demands and its goal is to minimize the maximum link utilization or MLU (TM = Traffic Matrix).

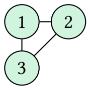
valuable information about the system and its components, which could have potentially helped with their search.

In this paper, we propose a *gray-box* approach (§3.2) that uses the *gradient* of the system to guide the search for adversarial inputs. Our insight is that if the components within the learning-enabled system satisfy certain properties, we can use the chain rule to combine the gradients from each individual component to estimate the end-to-end gradient of the entire system (see §3). We can then search in the direction of this gradient to find inputs that cause the learning-enabled system to underperform. Our method is faster and easier to use than white-box alternatives because it does not need an exact mode for each component and can be parallelized. It is also more effective than black-box local search methods.

We use DOTE [36] to illustrate our approach (§4). DOTE is a state-of-the-art traffic engineering system that replaces the optimization-based methods with a DNN to route traffic within a wide-area network [17, 19]. Our approach finds inputs that can cause DOTE to increase link utilization by over 6× compared to the optimal (§5). This suggests that using DOTE in production can cause unnecessary congestion, delays, and packet drops under certain demands. In contrast, the authors of DOTE [36] found that maximum link utilizations were within 1.05× of the optimal on the test data.

Our contributions are as follows:

- We show it is important to analyze the impact of DNNs as part of the entire systems in which they are deployed in order to evaluate their risk (see §2).
- We propose the idea for an analyzer that allows us to use gradients [14] to guide the search for bad inputs and discuss the further research we need to conduct to extend it to any learning-enabled system.
- We show the viability of this idea by applying it to DOTE and showing that it can perform significantly worse than what its authors reported.



Demand		Routing A		Routing B		Routing C	
src-dst	rate	path	rate	path	rate	path	rate
1-2	100	1-2	100	1-3-2	100	1-2	100
1-3	100	1-3	100	1-2-3	100	1-2-3	100
		MLU$_A$	**1**	MLU$_B$	**1**	MLU$_C$	**2**

Figure 3: Importance of analyzing the impact of DNNs on the final performance of learning-enabled systems. (left) topology with link capacities = 100. (right) a set of demands and three possible routings. Routing A and B use different split ratios but result in the same MLU.

2 Learning-enabled Traffic Engineering

Recent wide-area network traffic engineering (WAN TE) solutions incorporate DNNs into their computation pipelines to improve the speed or performance [3, 24, 26, 30, 36, 46].

For example, a state-of-the-art learning-enabled TE system, DOTE [36], (Figure 2) uses a DNN that takes the K most recent traffic matrices as input and predicts the traffic *split ratios* for the next epoch. These split ratios dictate how each source should split its traffic to each destination over a set of predetermined paths. DOTE uses a post-processor to ensure the DNN's outputs are feasible and meet network constraints (*e.g.*, the sum of a demand's split ratios should be 1).

Before using such a learning-enabled TE system, operators must ensure that it performs well – efficiently routes customer demands – under diverse network conditions and adapts to demand or topology changes [1, 34]. We argue that this is hard to do if we only consider the DNN in isolation.

Suppose an operator wants to minimize the maximum link utilization (MLU), which is a commonly used objective in WAN TE [17]. The split ratios (DNN's output) alone do not determine how DOTE impacts the MLU. In fact, two very different sets of split ratios may produce the same MLUs depending on the underlying demands (Figure 3). Therefore, we need to analyze the end-to-end pipeline instead of focusing on the DNN in isolation. This means we need to use these split ratios to route the demands and measure the MLU.

Many of the operators' questions about DOTE require an end-to-end analysis of the system's performance:

- How much can DOTE's MLU deviate from the optimal?
- What inputs cause DOTE to underperform?
- Are there inputs from the training data distribution that could cause DOTE to underperform?
- How does MLU of DOTE compare to another learning-enabled design, such as Teal?

The same observation applies to other recent work [26, 46] that uses similar pipelines but differs in the DNN architecture or the components involved.

We observe that we need fundamentally new techniques (compared to what exists today [13, 33]) to answer these questions and analyze the performance of learning-enabled pipelines. We discuss this assertion next.

3 End-to-End Performance Analysis

An end-to-end performance analyzer for a learning-enabled system must: (1) reason about the DNN's impact on the overall system performance and (2) find inputs that cause the learning-enabled system to underperform compared to an acceptable baseline (*e.g.,* the optimal).

We next discuss why today's performance analyzers cannot solve this problem.

3.1 Drawbacks of Existing Systems

White-box analyzers require describing the entirety of the learning-enabled system (which includes the DNN) and modeling it in a format that these tools support (either as an optimization [33] or in first-order logic [13, 32]). This requirement limits their scalability and expressiveness. These methods cannot:

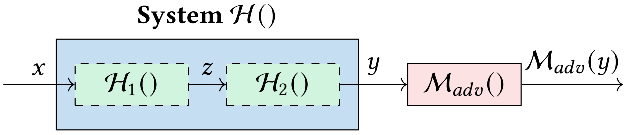
Analyze large and complex DNNs. Prior work that models DNNs in optimization or logic has limited ability to express complex architectures and does not scale well [10, 11, 21, 22, 27, 37, 43]. They only support small and specific neural network architectures with specific activations (*i.e.,* piece-wise linear activations such as ReLU). However, recent learning-enabled systems [26, 36, 46] use complex DNNs that fall outside of this category. It is difficult to extend existing tools to support these DNNs [10], and doing so requires approximations that adversely impact their ability to scale.

Jointly model all components within a system. It may be possible to efficiently model individual pieces of a learning-enabled pipeline. However, modeling them jointly requires faithfully capturing the dependencies between a diverse set of components. For instance, the outputs of one component are variables that serve as inputs to the next. These dependencies can result in non-convex, non-linear formulations, which are computationally hard to solve [33].

Black-box local search methods are an alternative to white-box heuristic analyzers [32]. They are general as they do not require any information about the system and its components [8, 23]. They iteratively search for adversarial inputs by selecting a new input in each iteration, executing the learning-enabled system and the optimal on these inputs, measuring their performance, and using the gap to guide future iterations. However, these methods get stuck in local optima and fail to find any useful adversarial input even for simple heuristics [32, 33].

3.2 A Gray-box Alternative

We explore the design of a gray-box tool to analyze the end-to-end performance of learning-enabled systems. Instead of requiring a detailed model of the entire system, we use partial information about its components to efficiently navigate the search space and find "adversarial" inputs that cause the system to underperform.



$$\nabla_x \mathcal{M}_{adv}(y_0) = \nabla_y \mathcal{M}_{adv}(y_0) \nabla_z \mathcal{H}_2(z_0) \nabla_x \mathcal{H}_1(x_0)$$

Figure 4: Using the chain rule to compute the end-to-end gradient of the system $\mathcal{H}$ as a function of the gradient of individual components $\mathcal{H}_1$ and $\mathcal{H}_2$.

Inspired by *gradient*-guided search algorithms [14], we propose using the *gradient* of the system to guide the search process: we search in the direction of the gradient to maximize the performance gap relative to the optimal.

Let $\mathcal{H}(x)$ denote the entire learning-enabled system, including the DNN(s). Suppose $\mathcal{M}_{adv}(\mathcal{H}(x))$ is a function that quantifies how "adversarial" the output of $\mathcal{H}(x)$ is. Our goal is to find inputs x that lead to large values of $\mathcal{M}_{adv}(\mathcal{H}(x))$. To find these adversarial inputs, we can move in the direction of the gradient of $\mathcal{M}_{adv}(\mathcal{H}(x))$:

$$x^{(i+1)} \leftarrow x^{(i)} + \alpha \nabla_x \mathcal{M}_{adv}(\mathcal{H}(x^{(i)})) \qquad (1)$$

where $x^{(i)}$ represents the input in iteration i, α is the step size, and $\nabla_x \mathcal{M}_{adv}(\mathcal{H}(x^{(i)}))$ is the gradient in iteration i. This iterative search method is guaranteed to converge to a global optimum [7] when the function $\mathcal{M}_{adv}(\mathcal{H}(.))$ is convex. Otherwise, it may find a local optimum.

Equation (1) still requires modeling a complex learning-enabled system as a closed form expression $\mathcal{H}(x)$, which goes against our initial goal for an easy-to-use gray-box solution. However, we can use this idea as the basis for a novel and practical search technique. We use the fact that DNNs are piecewise sub-differentiable[1]. If the other components within the learning-enabled system are also piecewise sub-differentiable, we can formulate our search in terms of the *combination* of the gradients of each of the individual components (see §6 for extension to other cases).

Formally, we can use the chain rule [14] (Figure 4) to compute the gradient of $\mathcal{M}_{adv}(\mathcal{H}(x))$. We first express $\mathcal{M}_{adv}(\mathcal{H}(x))$ as a composite function of the individual system components: $\mathcal{M}_{adv}(\mathcal{H}_n(\mathcal{H}_{n-1}(\ldots(\mathcal{H}_2(\mathcal{H}_1(x))$. Each function $\mathcal{H}_i$ represents a component of the system that takes the output of $\mathcal{H}_{i-1}$ as input and produces an output that feeds into $\mathcal{H}_{i+1}$. We can compute the gradient of each $\mathcal{H}_i$ with respect to its own input separately and use the chain rule to compute the gradient of $\mathcal{M}_{adv}(\mathcal{H}(x))$. Finally,

[1] They consist of components that are themselves individually differentiable (e.g., convolutional layers). These layers may be interspersed with other layers that are not differentiable such as ReLU activations.

we perturb the input in the direction of the gradient to increase $\mathcal{M}_{adv}$. This approach is mathematically equivalent to Equation (1).

We do not need an exact model of each function $\mathcal{H}_i$ to compute the gradients. We can either compute the gradient through its mathematical representation or compute it locally through samples of the function [39].

These techniques enable our gray-box solution as we do not need a detailed model of the entire system. Our solution has two important benefits: (1) following the gradient enables fast search and can help find bad inputs quickly, and (2) we can compute the gradient of each function in parallel, which allows us to speed up the search even further. These properties help us scale our analyzer beyond what existing tools are capable of (§5).

4 Gray-box Analysis of DOTE

We apply this technique to DOTE [36], a learning-enabled TE solution, to demonstrate its viability in practice. For this, we first need to define the function $\mathcal{M}_{adv}$, which measures how "adversarial" an output of DOTE is in terms of the end-to-end system performance (the final MLU in Figure 2).

The performance ratio. In [36], the efficacy of DOTE relative to the optimal is defined as the ratio between DOTE's MLU and that of the optimal. Accordingly, a natural definition for $\mathcal{M}_{adv}$ is the highest MLU ratio (called DOTE's *performance ratio*) for a given demand d:

$$\mathcal{M}_{adv}(d) \coloneqq \frac{\mathsf{MLU}_{\mathsf{DOTE}}(d)}{\mathsf{MLU}_{\mathsf{OPT}}(\mathsf{d})} = \max_{f} \frac{\mathsf{MLU}_{\mathsf{DOTE}}(d)}{\mathsf{MLU}(d,\ f)} \tag{2}$$

where our goal is to find split ratios f that result in the minimum possible MLU (the optimal split ratios) for each demand matrix d. The maximization in Equation (2) captures these split ratios by minimizing the denominator.

Equation (2) is a non-convex function of the demand d. This implies that we may end up converging to a local optimum if we use the gradient-based search method. Prior work [6] shows how we can rewrite this non-convex function as an equivalent convex objective to find the global optimum. It essentially constrains the set of demands to those the optimal TE solution can route without congestion (i.e., $\mathsf{MLU}(d, f) = 1$). The modified objective is:

$$\begin{aligned} &\mathcal{M}_{adv}(d) \coloneqq \mathsf{MLU}_{\mathsf{DOTE}}(\mathsf{d}) \\ &\text{over the space} \quad d \in \{d \mid \exists f:\ \mathsf{MLU}(d,\ \ f) = 1\} \end{aligned} \tag{3}$$

Note that the maximum possible MLU ratio from Equation (3) is equivalent to the one from Equation (2). This is because there is a linear relation between the MLU and the demands. Therefore, we can always normalize any demand from Equation (2) so that it falls in the feasibility space of Equation (3). The normalization step does not impact the optimal split ratios and only reduces the optimal MLU by the normalization factor. If DOTE's split ratios remain the same, we can guarantee that the performance ratio also stays the same. We discuss later how to deal with cases where this does not hold.

Using Lagrangian relaxation. We need to express the end-to-end loss function as an unconstrained optimization before computing the gradient. A standard technique to do this is called Lagrangian relaxation:

$$\min_{\lambda} \max_{d,\ f} \mathcal{L}_{final}(d, f, \lambda) \ \coloneqq\ \mathcal{M}_{adv}(d) + \lambda(\mathsf{MLU}(d,\ \ f) - 1) \tag{4}$$

where the Lagrange multiplier λ is a penalty term that encourages the optimizer toward values of d and f such that $\mathsf{MLU}(d, f) = 1$ (Equation (3) is feasible).

Equation (4) is an instance of a minimax optimization where we minimize over λ (outer) and maximize over d and f (inner). We use multi-step gradient descent ascent [35] to solve this problem iteratively. In each iteration, we first solve the inner maximization problem by taking T gradient ascent steps over d and f (T is configurable). We then use the updated values of d and f to estimate the gradient of the outer minimization and take a gradient descent step over λ:

$$\begin{aligned} d^{(i+1)} &\leftarrow d^{(i)} + \alpha_d \nabla_d \mathcal{L}_{final}(d^{(i)}, f^{(i)}, \lambda^{(i)}) \\ f^{(i+1)} &\leftarrow f^{(i)} + \alpha_f \nabla_f \mathcal{L}_{final}(d^{(i)}, f^{(i)}, \lambda^{(i)}) \\ \lambda^{(i+1)} &\leftarrow \lambda^{(i)} - \alpha_\lambda \nabla_\lambda \mathcal{L}_{final}(d^{(i)}, f^{(i)}, \lambda^{(i)}) \end{aligned} \tag{5}$$

where α_d is a hyper-parameter that determines step sizes in the demand search space (similarly for f and λ) and $\nabla_d \mathcal{L}_{final}$ denotes the gradient with respect to the demands. We determine these gradients using the chain rule (§3).

Other TE Objectives. We can extend our analysis to other traffic engineering objectives, such as those that maximize total flow [1] or maximize concurrent flow [36]. To do so, we need to describe the end-to-end performance function and compute the gradients.

The main challenge is that the linear relationship between the performance function and demands may not hold for other objectives, such as total flow. Therefore, we cannot replace the non-convex function in Equation (2) with the convex alternative in Equation (3) without losing optimality guarantees. We address this issue by changing the feasible space of Equation (3) to include demands that the optimal would achieve a given performance P ($\{d \mid \exists f:\ \mathsf{OPT}(d,\ \ f) = P\}$). We then search for the value of P that results in the largest performance ratio. Our method is fast (§5), so we can run it multiple times. For MLU, solving for $P = 1$ is sufficient.

The optimal formulation may contain additional constraints, such as capacity constraints [17, 34]. We can relax them using the Lagrangian method.

Method	**Discovered** $\frac{\text{MLU}_{\text{DOTE}}}{\text{MLU}_{\text{OPT}}}$	**Runtime**
DOTE's test set	1.05×	–
Random Search	1.22×	25 seconds
MetaOpt	–	6 hours
Gradient-based	**6×**	**50 seconds**

Table 1: Our proposed method discovers an MLU ratio of 6× for DOTE-Hist while others fail to produce meaningful analysis.

Method	**Discovered** $\frac{\text{MLU}_{\text{DOTE}}}{\text{MLU}_{\text{OPT}}}$	**Runtime**
DOTE's test set	1.05×	–
Random Search	1.25×	20 seconds
MetaOpt	–	6 hours
Gradient-based	**3.47×**	**54 seconds**

Table 2: Our proposed method discovers an MLU ratio of 3.47× for DOTE-curr while others fail to produce meaningful analysis.

5 Performance Analysis of DOTE: Results

We evaluate DOTE using our gray-box performance analyzer and show it finds substantially worse adversarial inputs than a white-box tool and a straw-man black-box search algorithm. We also show it is fast and uncovers the performance ratio in a few minutes.

We use the K-shortest paths algorithm [48] to configure the set of available paths for each demand ($K = 4$). We constrain the demands to be below a maximum value (the average link capacity) to ensure they are realistic. We evaluate two different versions of DOTE:

- DOTE-Hist takes the last 12 traffic demands as input and estimates the split ratios for the next epoch (see [36]).
- DOTE-Curr is a modified version of DOTE that takes the current traffic as input. This is similar to other learning-enabled traffic engineering systems such as Teal [46].

We report DOTE's performance ratio (§4) based on its MLU on the Abilene topology [40] and repeat each experiment 5 times to ensure the results are stable. We run each method for 6 hours or until they terminate. We report the runtime as the earliest point at which the method identified a gap and was unable to make further improvements. We run all experiments on an AMD Opteron 2.4 GHz CPU with 24 cores and 64GB of memory. We also set $\alpha_d = \alpha_\lambda = \alpha_f = 0.01$ and $T = 1$ (unless mentioned otherwise).

Baselines. We compare our solution to (1) the performance of DOTE on its test set, (2) Random search (a black-box method), and (3) MetaOpt [33] (a white-box approach). We extended MetaOpt's code to support DNNs and all the other components in DOTE's pipeline. We had to replace DOTE's non-linear activation function with a piece-wise linear alternative to be able to use MetaOpt.

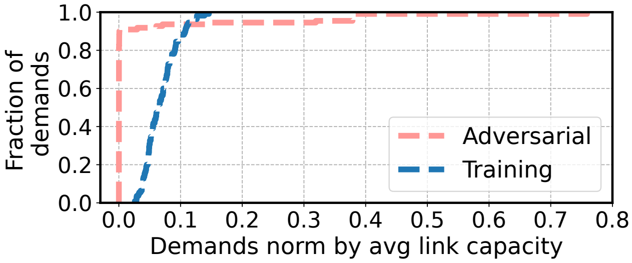


Figure 5: Comparing the demand sizes in the adversarial input from our gray-box approach with a representative sample from DOTE's training data.

DOTE exhibits higher MLU relative to the optimal (Tables 1 and 2). Using both variants of DOTE (DOTE-hist and DOTE-curr) can lead to a substantially higher MLU compared to the optimal (6× and 3.47×, respectively). This is much larger than the gap DOTE's authors discovered by evaluating it on the test data (they found DOTE is always within 1.05× of the optimal).

DOTE-curr has a smaller gap compared to DOTE-hist because DOTE-hist attempts to estimate the split ratios from the past demands, which can fail if the traffic distribution suddenly changes. However, DOTE-curr is aware of the traffic in the next epoch.

Our analyzer outperforms other baselines. It efficiently finds adversarial inputs that lead to significantly worse performance compared to black-box random search methods (MLU ratios 2.78 to 5× larger). Our runtimes are around 1 minute, whereas MetaOpt could not find any MLU ratio even after 6 hours. We need to jointly model the DNN and all the other components in optimization to use MetaOpt. This increases the complexity of the optimization solver and reduces its scalability.

The adversarial demands (Figure 5). We find the adversarial demands from our tool are different from the demands in DOTE's training set. The training data contains demands where most pairs exchange small traffic. In contrast, only a few pairs exchange the majority of the traffic in the adversarial examples.

It is important to ensure that DOTE is resilient to such changes in demand before we deploy it in production. These changes may occur in practice, such as when a fiber cut happens and causes a shift in the traffic distribution. We can use the adversarial samples from our tool and add them to DOTE's training set to improve its performance for such cases. If the operator is only interested in adversarial inputs that typically occur in practice, we can also include additional constraints to ensure that our tool generates inputs from these specific distributions (see §6).

step size α_d	Discovered $\frac{\text{MLU}_{\text{DOTE}}}{\text{MLU}_{\text{OPT}}}$	Runtime
0.01	3.47×	54 seconds
0.005	3.47×	73 seconds
0.05	3.46×	44 seconds

Table 3: Sensitivity of the gradient-based approach to the hyper-parameter α_d (Equation (5)). We set $\alpha_\lambda = 0.01$ and $\alpha_f = 0.01$.

Sensitivity Analysis (Table 3). The step size in Equation (5) is a hyper-parameter in our approach. It introduces a trade-off: smaller step sizes allow us to find larger gaps because they traverse the space in smaller granularities and can get closer to the global optimal. However, they take longer to converge. Larger step sizes may find a slightly lower performance gap because they jump around the optimal point.

6 Extensions and Generalizations

We show our idea for a gray-box performance analyzer based on gradient-based search is viable. There are several possible directions that require more research:

Comparing to other learning-enabled systems. In §4, we compare DOTE to the optimal solution. We can also compare it to other learning-enabled TE pipelines [26, 46]. To do so, we have to modify the performance function in Equation (2) and replace the optimal MLU with that of a given baseline, such as Teal.

Constraining bad inputs. Our method searches for adversarial inputs over the entire input space by default, but we can also limit it to search over *realistic inputs*. In the context of traffic engineering, these are the demands that are sparse and exhibit locality [33]. We can encode these as additional constraints on the feasible space in Equation (3). Then, we can apply the Lagrangian relaxation (§4) to transform the constrained optimization into an unconstrained one and conduct the search using the gradient.

Beyond single adversarial example. We can also extend our gradient-based search method to: (1) learn and generate a corpus of examples that can cause the learning-enabled systems to underperform *in one shot* (as opposed to just a single bad input), and (2) find adversarial examples that belong to a specific distribution (*e.g.,* those that have the same distribution as the training data). We can use the output of the former and augment the training data with the adversarial samples to improve the DNN. The latter can help operators find the worst-typical performance of the DNN.

To enable these use-cases, we can potentially use the system's gradient to train two neural networks: a generator and a discriminator (as in GANs [15]). The generator learns to generate plausible inputs that would cause the learning-enabled system to underperform. The discriminator determines whether the inputs from the generator are from the specified distribution. These two components work together to create a set of realistic adversarial inputs from a target distribution (see [15]).

Mechanisms that approximate non-differentiable components. Some learning-enabled systems [24, 30, 47] may introduce objectives that are not (sub)differentiable. If we approximate non-differentiable components in the learning-enabled systems with differentiable functions, we can still compute the gradient, apply the chain rule, and conduct the search. Many functions enable such approximations. A DNN is one such example that can learn to approximate a function from samples of its outputs [38]. We can integrate the training of this DNN into our search (Equation (4)) by adding a regularization term that minimizes the difference between the true output of the non-differentiable component ($\bar{h}$) and the output of the DNN that approximates it ($f_\theta(.)$):

$$\min_{\theta} \ \mathcal{L}_{diff} \coloneqq ||f_\theta(x) - \bar{h}||_2^2$$

Another similar option is to use a Gaussian process [39]. Future research should determine what approximations are most effective.

Partitioning the performance analysis. In some cases, it can be hard to approximate the component because of its inherent complexity [38]. Instead, we can explore techniques to break down the learning-enabled system into smaller sub-systems and use a different approach to analyze each of these sub-systems.

Consider a learning-enabled pipeline with the following sub-systems in order: $\mathcal{H}_1, \ldots, \mathcal{H}_{n-1}$, and $\mathcal{H}_n$. One potential idea is to start from the last sub-system $\mathcal{H}_n$ and find the inputs to this function that constitute its "adversarial space" (worst-case scenarios). Once we find this adversarial space, we move one step back: we analyze $\mathcal{H}_{n-1}$ and find how to push it to produce outputs in $\mathcal{H}_n$'s adversarial space (*i.e.,* what we found in the previous step). We then continue to iterate in this manner until we analyze the entire system and produce the set of inputs that cause the overall system to underperform. In other words, we move *backwards* stage-by-stage through each component until we find inputs to the learning-enabled system that cause the entire system to underperform.

We can analyze the (approximately) differentiable sub-systems using our gradient search method (§4). We can model other sub-systems as an optimization [16] or in logic [9] and use existing performance analyzers [33]. This approach scales better since we can model each sub-system more efficiently in isolation than jointly modeling all of them.

The key to our partitioning approach is the notion of an *adversarial space* for any given sub-system. This is the space of inputs *for that subsystem* that drives it to perform poorly. However, existing tools [13, 33] only generate a single adversarial instance. We can adopt prior work that explore ways to extrapolate from these instances into an adversarial space [20].

Improving robustness of learning-enabled systems. We can potentially use the adversarial examples from our gradient-based search method to improve the learning-enabled system. One way to do this is to add these examples to the DNN's training data but we need to ensure that this does not adversely impact the DNN's average performance.

Beyond learning-enabled system. Although we have focused our paper on learning-enabled systems, our approach is more broadly applicable to the performance analysis of any system with (approximately) piecewise sub-differentiable components. As such, it can augment the suite of performance analyzers currently in our toolkit [13, 33]. To use our approach, we only have to model the objective of the system and estimate the gradient of its components. This is simpler that *exactly* and *jointly* modeling all the components as in other tools.

7 Related Work

Performance Analysis Tools. Prior work [2, 4, 5, 32, 33] develops tools to analyze the performance of heuristics and network control algorithms. MetaOpt [32, 33] formulates the heuristic as an optimization and finds adversarial inputs using Stackleberg games and Bilevel optimization. Virelay [13] models resource allocation heuristics in formal methods and uses SMT solvers [9]. Other work [2, 4, 5] develop custom tools to answer performance queries in specific domains (*e.g.,* congestion control). These tools need an exact model of the entire system either as optimization or in first-order logic, which limits their ability to analyze learning-enabled systems (§3.1).

Adversarial Attacks for Neural Networks. [18, 25, 28, 29] focus on adversarial attacks for classification tasks. Their goal is to measure the sensitivity of neural networks and find the minimum perturbation to an existing input that would cause the neural network's classification to change. This is different from our goal: we consider neural networks as part of an end-to-end system and want to find an input from scratch that would cause the system to underperform.

Safety of DNNs [11, 21, 22, 27, 37, 43] ensure DNNs match user-define safety properties. A few focus on DNNs in networks and systems [10, 41, 42]. However, these solutions (1) evaluate safety of the DNN in isolation and (2) can only analyze specific DNN architectures (*e.g.,* feed-forward neural networks with piece-wise linear activations).

Other work [12, 31] focuses on finding malicious inputs for network control algorithms. [45] improves the robustness of RL policies by finding difficult environments, but they can only compare the RL policy with static rule-based policies. Our work focuses on DNN-based systems and enables comparison with complex baselines (including other DNN-based systems or the optimal).

8 Conclusion

We discuss the importance of analyzing DNNs as part of the entire system in which they are deployed. To evaluate these learning-enabled systems, we propose a gray-box performance analyzer that uses gradients to identify bad inputs. We demonstrate that this approach overcomes the limitations of white-box and black-box local search methods. We also outline future research to extend and generalize this gray-box method.

Acknowledgments. We thank the anonymous reviewers for their insightful comments. This material is based upon work supported in part by the NSF grant No. CNS-1901523, ISF grant No. 3481/24, and BSF grant No. 2023663.

References

[1] Firas Abuzaid, Srikanth Kandula, Behnaz Arzani, Ishai Menache, Matei Zaharia, and Peter Bailis. 2021. Contracting Wide-area Network Topologies to Solve Flow Problems Quickly. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 175–200. https://www.usenix.org/conference/nsdi21/presentation/abuzaid

[2] Anup Agarwal, Venkat Arun, Devdeep Ray, Ruben Martins, and Srinivasan Seshan. 2024. Towards provably performant congestion control. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 951–978. https://www.usenix.org/conference/nsdi24/presentation/agarwal-anup

[3] Abd AlRhman AlQiam, Yuanjun Yao, Zhaodong Wang, Satyajeet Singh Ahuja, Ying Zhang, Sanjay G. Rao, Bruno Ribeiro, and Mohit Tawarmalani. 2024. Transferable Neural WAN TE for Changing Topologies. In *Proceedings of the ACM SIGCOMM 2024 Conference*.

[4] Mina Tahmasbi Arashloo, Ryan Beckett, and Rachit Agarwal. 2023. Formal Methods for Network Performance Analysis. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 645–661. https://www.usenix.org/conference/nsdi23/presentation/tahmasbi

[5] Venkat Arun, Mina Tahmasbi Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. [n. d.]. Toward formally verifying congestion control behavior. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*.

[6] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. 2003. Optimal oblivious routing in polynomial time. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing (STOC '03)*. Association for Computing Machinery.

[7] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization.* Cambridge university press.

[8] Lawrence Davis. 1991. Bit-Climbing, Representational Bias, and Test Suite Design. In *Proceedings of the 4th International Conference on Genetic Algorithms*, Richard K. Belew and Lashon B. Booker (Eds.).

[9] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340.

[10] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. 2021. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 305–318. https://doi.org/10.1145/3452296.3472936

[11] Matteo Fischetti and Jason Jo. 2018. Deep neural networks and mixed integer linear optimization. *Constraints* 23, 3 (2018), 296–309.

[12] Tomer Gilad, Nathan H. Jay, Michael Shnaiderman, Brighten Godfrey, and Michael Schapira. 2019. Robustifying Network Protocols with Adversarial Examples. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19)*.

[13] Saksham Goel, Benjamin Mikek, Jehad Aly, Venkat Arun, Ahmed Saeed, and Aditya Akella. 2023. A Performance Verification Methodology for Resource Allocation Heuristics. https://api.semanticscholar.org/CorpusID:255595504

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

[16] Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[17] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. *SIGCOMM Comput. Commun. Rev.* 43, 4 (aug 2013), 15–26. https://doi.org/10.1145/2534169.2486012

[18] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[19] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: experience with a globally-deployed software defined wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (Hong Kong, China) *(SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/2486001.2486019

[20] Pantea Karimi*, Solal Pirelli*, Siva Kesava Reddy Kakarla, Ryan Beckett, Santiago Segarra, Beibin Li, Pooria Namyar, and Behnaz Arzani. 2024. Towards Safer Heuristics With XPlain. *arXiv preprint arXiv:2410.15086*. https://arxiv.org/abs/2410.15086

[21] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2021. Reluplex: a Calculus for Reasoning about Deep Neural Networks. *Formal Methods in System Design* (2021).

[22] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle D. Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *International Conference on Computer Aided Verification*.

[23] Scott Kirkpatrick, D. Gelatt Jr., and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* (1983).

[24] Jitendra Kumar and Ashutosh Kumar Singh. 2019. Cloud Resource Demand Prediction using Differential Evolution based Learning. In *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. 1–5. https://doi.org/10.1109/ICSCC.2019.8843680

[25] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. 2018. Adversarial examples in the physical world. In *Artificial intelligence safety and security*. Chapman and Hall/CRC, 99–112.

[26] Ximeng Liu, Shizhen Zhao, and Yong Cui. 2024. FIGRET: Fine-Grained Robustness-Enhanced Traffic Engineering. *ACM SIGCOMM* (2024).

[27] Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward ReLU neural networks. (2017). http://arxiv.org/abs/1706.07351

[28] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. 2020. Towards more practical adversarial attacks on graph neural networks. *Advances in neural information processing systems* 33 (2020), 4756–4766.

[29] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[30] Tanwi Mallick, Mariam Kiran, Bashir Mohammed, and Prasanna Balaprakash. 2020. Dynamic Graph Neural Network for Traffic Forecasting in Wide Area Networks. *2020 IEEE International Conference on Big Data (Big Data)* (2020), 1–10.

[31] Roland Meier, Thomas Holterbach, Stephan Keck, Matthias Stähli, Vincent Lenders, Ankit Singla, and Laurent Vanbever. 2019. (Self) Driving Under the Influence: Intoxicating Adversarial Network Inputs. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (Princeton, NJ, USA) *(HotNets '19)*. 34–42.

[32] Pooria Namyar, Behnaz Arzani, Ryan Beckett, Santiago Segarra, Himanshu Raj, and Srikanth Kandula. 2022. Minding the gap between fast heuristics and their optimal counterparts. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks (HotNets '22)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3563766.3564102

[33] Pooria Namyar, Behnaz Arzani, Ryan Beckett, Santiago Segarra, Himanshu Raj, Umesh Krishnaswamy, Ramesh Govindan, and Srikanth Kandula. 2024. Finding Adversarial Inputs for Heuristics using Multilevel Optimization. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 927–949. https://www.usenix.org/conference/nsdi24/presentation/namyar-finding

[34] Pooria Namyar, Behnaz Arzani, Srikanth Kandula, Santiago Segarra, Daniel Crankshaw, Umesh Krishnaswamy, Ramesh Govindan, and Himanshu Raj. 2024. Solving Max-Min Fair Resource Allocations Quickly on Large Graphs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1937–1958. https://www.usenix.org/conference/nsdi24/presentation/namyar-solving

[35] Maher Nouiehed, Maziar Sanjabi, Tianjian Huang, Jason D Lee, and Meisam Razaviyayn. 2019. Solving a class of non-convex min-max games using iterative first order methods. *Advances in Neural Information Processing Systems* 32 (2019).

[36] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. 2023. DOTE: Rethinking (Predictive) WAN Traffic Engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1557–1581. https://www.usenix.org/conference/nsdi23/presentation/perry

[37] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Semidefinite relaxations for certifying robustness to adversarial examples. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS'18)*.

[38] Jason Ramapuram and Russ Webb. 2020. Improving Discrete Latent Representations With Differentiable Approximation Bridges. In *2020 International Joint Conference on Neural Networks (IJCNN)*. 1–10. https://doi.org/10.1109/IJCNN48605.2020.9207581

[39] Eric Schulz, Maarten Speekenbrink, and Andreas Krause. 2018. A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of mathematical psychology* 85 (2018), 1–16.

[40] Stanford University IT. 2015. Abilene Core Topology. https://uit.stanford.edu/service/network/internet2/abilene

[41] Cheng Tan, Changliu Liu, Zhihao Jia, and Tianhao Wei. 2023. Building Verified Neural Networks for Computer Systems with Ouroboros. In *Proceedings of Machine Learning and Systems*.

[42] Cheng Tan, Yibo Zhu, and Chuanxiong Guo. 2021. Building verified neural networks with specifications for systems. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems* (Hong Kong, China) *(APSys '21)*. 42–47.

[43] Vincent Tjeng, Kai Xiao, and Russ Tedrake. [n. d.]. Evaluating Robustness of Neural Networks with Mixed Integer Programming. *ICLR 2019* ([n. d.]).

[44] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. 2017. Learning to Route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets '17)*. Association for Computing Machinery, New York, NY, USA.

[45] Zhengxu Xia, Yajie Zhou, Francis Y. Yan, and Junchen Jiang. 2022. Genet: automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*.

[46] Zhiying Xu, Francis Y. Yan, Rachee Singh, Justin T. Chiu, Alexander M. Rush, and Minlan Yu. 2023. Teal: Learning-Accelerated Optimization of WAN Traffic Engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 378–393. https://doi.org/10.1145/3603269.3604857

[47] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 495–511. https://www.usenix.org/conference/nsdi20/presentation/yan

[48] Jin Y. Yen. 1971. Finding the K Shortest Loopless Paths in a Network. *Management Science* 17, 11 (1971), 712–716.