SPECIFICATION GENERATION FOR NEURAL NETWORKS IN SYSTEMS

Isha Chaudhary [1] **Shuyi Lin** [2] **Cheng Tan** [2] **Gagandeep Singh** [1 3]

ABSTRACT

Specifications — precise mathematical representations of correct domain-specific behaviors — are crucial to guarantee the trustworthiness of computer systems. With the increasing development of neural networks as computer system components, specifications gain more importance as they can be used to regulate the behaviors of these black-box models. Traditionally, specifications are designed by domain experts based on their intuition of correct behavior. However, this is labor-intensive and hence not a scalable approach as computer system applications diversify. We hypothesize that the traditional (aka reference) algorithms that neural networks replace for higher performance can act as effective proxies for correct behaviors of the models, when available. This is because they have been used and tested for long enough to encode several aspects of the trustworthy/correct behaviors in the underlying domain. Driven by our hypothesis, we develop a novel automated framework, SpecTRA (**Spec**ifications from **T**rustworthy **R**eference **A**lgorithms) to generate specifications for neural networks using references. We formulate specification generation as an optimization problem and solve it with observations of reference behaviors. SpecTRA clusters similar observations into compact specifications. We present specifications generated by SpecTRA for neural networks in adaptive bit rate and congestion control algorithms. Our specifications show evidence of being correct and matching intuition. Moreover, we use our specifications to show several unknown vulnerabilities of the SOTA models for computer systems.

1 INTRODUCTION

Neural Networks (NNs) have recently found numerous applications as integral components in computer systems (Jay et al., 2019; Mao et al., 2017; 2019; Mendis et al., 2019). For example, they have been applied to enhance video streaming quality, congestion control, database query optimization, indexing, scheduling, and various other system tasks. Compared to traditional heuristic-based approaches, NNs reduce development overheads and offer improved average performance (Kraska, 2021).

Despite recent advancements, skepticism remains about the practicality of NNs in computer systems. A key concern is that user-facing systems, like video streaming, require high standards of performance and reliability in dynamic environments. Although NNs often surpass traditional methods in average performance, they are fundamentally opaque (Chaudhary et al., 2024) and lack guarantees.

Indeed, researchers have identified numerous counterintuitive behaviors of these NNs in computer systems. For example, Eliyahu et al. (2021) observed that Aurora (Jay et al., 2019), a congestion control system, in certain conditions, would repeatedly decrease its sending rate, ultimately reaching and maintaining the minimal rate despite excellent network conditions. In another example, Meng et al. (2020, §6.3) "debugged" Pensieve (Mao et al., 2017), a bitrate controller for video streaming, which systematically avoided two specific bitrates (1200 and 2850kbps); they introduced these bitrates back to enhance performance.

Note that these observations are not unique; similar behaviors have been extensively studied in other domains, such as vision (Szegedy et al., 2014; Liu et al., 2023; Athalye et al., 2018). For instance, in image classification, researchers have identified adversarial images that can mislead vision models, such as altering a stop sign to resemble a speed limit sign (Eykholt et al., 2018). To address this issue, robustness specifications have been introduced—requiring that, despite noise, the image should still be classified correctly (Gehr et al., 2018; Balunovic et al., 2019; Mirman et al., 2020).

However, unlike vision tasks, where some notions of correctness exist (Athalye et al., 2018; Yang et al., 2023), determining correctness in system applications is more challenging. Typically, manually-designed specifications are hard to design, error-prone, and limited to some corner-cases. Consider the example of a video bitrate controller: given the current network conditions and buffered video frames, it is difficult to manually label bitrate choices as "incorrect". Prior work (Eliyahu et al., 2021) has introduced specifications based on extreme cases, excluding several desirable trustworthy behaviors as we observe in our experiments. Furthermore, the authors of NN4SysBench (He et al., 2022), a benchmark suite for NNs in systems, acknowledge that

[1]University of Illinois, Urbana-Champaign [2]Northeastern University [3]VMWare Research. Correspondence to: Isha Chaudhary <isha4@illinois.edu>.

Preprint. Under review.

their specifications are inherently subjective. Our aim is to mitigate these drawbacks of manually-designed specifications with automated specification generation for NNs in computer systems. We desire the specifications to be expressive, containing multiple aspects of correct/trustworthy behaviors in the given applications. Thus, we investigate:

How can we automatically generate specifications encoding several trustworthy behaviors for NNs in computer systems?

Main idea. We observe that a unique characteristic of computer systems is the presence of traditionally-used rule-based algorithms and heuristics—referred to as *references*. Although these references may not match the performance of neural networks, they are considered reliable, having been crafted according to domain experts' understanding of correct behavior and rigorously tested in production environments. This forms an analogy: references in systems serve a similar role as, for example, human perception does in image classification. By collecting the outputs of these references across various inputs and environments, we can define specifications for their neural network counterparts.

Implementing this idea presents three key challenges. First, multiple traditional implementations often exist for the same application; for instance, model predictive control (Yin et al., 2015) and buffer-based algorithms (Huang et al., 2014) are both widely used adaptive bitrate algorithms. These implementations may exhibit divergent behaviors on the same input, making it unclear how to consolidate their feedback into specifications for NNs. Second, the set of specifications derived from references is not unique, and some specifications are more useful than others. Thus, we need to design criteria to identify *high-quality* specifications. Finally, synthesizing specifications is generally expensive and error-prone (Wen et al., 2024; Albarghouthi et al., 2016; Bastani et al., 2015). We need an efficient algorithm capable of generating specifications with the desired traits.

Our approach. We propose an automated specification generation method for system domains where references are available. Our specifications are based on pre and post-conditions consisting of constraints on the neural network inputs and outputs respectively (Hoare, 1969). We specify that if the network input satisfies the precondition, its output must satisfy the postcondition. The postconditions are constructed by combining the behaviors of multiple references.

We formalize the desired traits that specifications should satisfy and pose their generation as an optimization problem (§ 3). We design an effective algorithm, *SpecTRA* to solve the optimization problem using clustering. Our algorithm assumes access to offline observations (i.e., we do not use the source code or make custom queries to get observations of the behaviors of references), therefore, our algorithm can handle references that have complicated implementations, whose source code is unavailable or inference is either impossible or expensive. We generate expressive and useful specifications for NNs for two challenging and practically important applications: adaptive bit rate streaming (Mao et al., 2017) and congestion control (Jay et al., 2019).

Solution scope. As the first step towards automatically generated specifications for NNs in computer systems, the specifications learned from references serve as guidance rather than strict requirements. In particular, some behaviors in reference implementations should be emulated by NNs, while others should be avoided, allowing NNs to potentially exceed the performance of conventional approaches—achieving this balance is essential. In addition, behavioral expectations are not absolute given the dynamically-varying environments in which computer systems are operated; and similarly specifications are *likely* encodings of the notion of trustworthiness in the given domain, as they are generated from observations of the references operated in these environments. Nonetheless, the specifications are useful for testing and verification (§5) and can serve as concise, interpretable descriptions of desirable behaviors in the chosen domain. The specifications can also be used to compare different NNs; a network that satisfies the specification while achieving high performance is more aligned with developer expectations than one that does not.

Contributions. Our main contributions are:

1. We formalize generating expressive and useful specifications from references as an optimization problem.

2. We develop an automated specification generation algorithm, called SpecTRA[1] using our formalism, that can handle complicated and closed-sourced references. SpecTRA leverages clustering to identify common behaviors across observations from the references. Code is available at `https://github.com/uiuc-focal-lab/spectra`.

3. We create specifications for NNs in Adaptive Bitrate and Congestion Control applications. We empirically demonstrate (§5) the high quality of our specifications. We also use SpecTRA's specifications to identify previously unknown vulnerabilities of the tested NNs.

SpecTRA is the first step towards formally verified NNs in computer systems. As SpecTRA's specifications are generated from references that are deemed trustworthy by domain experts, checking and enforcing NNs to adhere to the specifications can enhance the reliability of these models and encourage their practical deployment.

[1] **Spec**ifications from **T**rustworthy **R**eference **A**lgorithms

2 BACKGROUND

We demonstrate specifications for 2 applications — Adaptive Bit Rate (ABR) video streaming (Sani et al., 2017) and Network Congestion Control (CC) (Jiang et al., 2020).

Adaptive Bitrate. Adaptive Bit Rate (ABR) algorithms are used to optimize the bit rate for streaming videos from servers to clients such that the Quality of Experience, QoE (Balachandran et al., 2013) for the users is maximized. Quality of experience is typically determined by the bit rate of video chunks (higher is better), and the startup time, rebuffering time, and bitrate variations (lower is better). An ABR algorithm observes the video streaming system's state consisting of buffer size and video chunk download time, observed throughput, size of the next video chunk at all possible bit rates, etc. to determine the bit rate at which the next video chunk should be fetched. Pensieve (Mao et al., 2017) is a popular neural network (NN) used for ABR. It is trained using reinforcement learning (RL) to maximize the QoE. We provide Pensieve's architectural details in Appendix A.

Congestion control. Congestion control is a regulatory process that determines the packet sending rate across a given network at any time, to maximize the network throughput (packets sent over the network) and minimize latency and packet loss (Jay et al., 2019). Congestion control algorithms use latency gradient (Dong et al., 2018) and latency ratio (Winstein et al., 2013) as input features and output the change in sending rate for the next time step. Aurora (Jay et al., 2019) is a popular NN-based solution for congestion control, trained with RL (architectural details in Appendix B). The reward here is a combination of the throughput, latency, and packet loss observed in the system.

3 FORMALIZING SPECIFICATIONS FROM REFERENCE ALGORITHMS

While neural networks attempt to maximize aggregate performance, we develop a set of specifications Ψ for individual inferences to satisfy for greater trust. Let $\mathcal{X} \subseteq \mathbb{R}^m$ be the sets of all possible m-dimensional ($m > 0$) inputs to the neural network for which we want to generate Ψ. Let $\mathcal{Y} = [1, \ldots, k]$ be the (discrete) set of all possible outputs of the neural network, where $1 < k < \infty$. Such instances with finite output sets are fairly common, e.g., in neural-network classifiers (Zhang, 2000; Deng, 2012) and RL agents over finite action spaces such as Pensieve (Mao et al., 2017). If that is not the case, we discretize $\mathcal{Y}$ when possible. For example, the output of the Aurora congestion control model (Jay et al., 2019) is continuous-valued and we generate specifications for it. We provide details of the output discretization for Aurora's specifications in §5.1. Each specification $\mathcal{S} \in \Psi$ describes the desirable behavior over a set of possible inputs called a *precondition*, denoted by the Boolean predicate $\varphi_{\mathcal{S}}$ that evaluates to true for inputs in the precondition. $\mathcal{S}$ mandates that for all inputs $x \in \mathcal{X}$ such that $\varphi_{\mathcal{S}}(x)$, the output $y \in \mathcal{Y}$ should follow a *postcondition*, denoted by the Boolean predicate $\psi_{\mathcal{S}}$, i.e., $\mathcal{S}(x, y) \triangleq \varphi_{\mathcal{S}}(x) \implies \psi_{\mathcal{S}}(y)$. The specifications in Ψ are considered to be in conjunction, i.e., the overall desirable behavior is $\bigwedge_{\mathcal{S} \in \Psi} \mathcal{S}$.

We leverage the behavior of q (> 0) traditional algorithms, aka references $\mathcal{R}_1, \ldots \mathcal{R}_q$ to determine Ψ. We use multiple references to avoid Ψ that overfit the behavior of one reference and contain its suboptimal behaviors. Specifically, we consider inputs $x \in \mathcal{X}$ for which the combined set of outputs from all the references $\bigcup_{j \in [q]} \mathcal{R}_j(x)$ is non-trivial, i.e., excludes some possible outputs from $\mathcal{Y}$. For other inputs, the references are not selective and do not provide useful information about the desirable behaviors in the target domain. Hence, they are not used to form the specifications. We use γ, the mapping between such x and their corresponding outputs from references $\bigcup_{j \in [q]} \mathcal{R}_j(x)$, as the *interesting behaviors* of references (Definition 1), which are used for generating specifications Ψ. We hypothesize and empirically show that if a neural network's output matches that of any reference for inputs in the interesting behaviors, then it will be more trustworthy.

Definition 1. *(Interesting behaviors of references). Interesting behaviors γ contain inputs $x \in \mathcal{X}$ for which all references $\mathcal{R}_1, \ldots, \mathcal{R}_q$ collectively lead to a non-trivial output set $y \subset \mathcal{Y}$, which is a strict subset of all possible outputs.*

$$\gamma \triangleq \left\{ (x, y) \mid x \in \mathcal{X} \wedge y = \bigcup_{j \in [q]} \mathcal{R}_j(x) \wedge y \subset \mathcal{Y} \right\}$$

We use γ_x to denote the inputs x in the interesting behaviors γ. A lookup table mapping γ_x to the corresponding outputs from the references is an exact set of specifications. However, it is not amenable to downstream applications such as verification of the neural networks as it can potentially consist of uncountably many entries. The lookup table may not have finite representations in a general case. Therefore, we combine several $x \in \gamma_x$ into a single concise representation $\varphi_{\mathcal{S}}$ via overapproximation in each of our specifications $\mathcal{S}$ and map it to $\psi_{\mathcal{S}}$ which captures the permissible output of x. The downside of simple and concise representations is that they can potentially introduce errors by restricting the outputs of the additional inputs $\hat{x} \notin \gamma_x$ that satisfy $\varphi_{\mathcal{S}}$ to $\psi_{\mathcal{S}} \subset \mathcal{Y}$. We attempt to minimize such overapproximation errors when developing our specifications.

Typically, concise representations consist of polyhedral constraints on permissible inputs, each of which is mapped to a common set of outputs (Brix et al., 2023). The simplest and widely used polyhedral representation consists of interval constraints. It leads to easily interpretable specifications. Hence, our specifications are defined using intervals that

overapproximate interesting behaviors.

Preconditions. The precondition of each specification consists of intervals over each dimension of the input space. Their canonical form is $\varphi_{\mathcal{S}}(x) = \forall i \in \{1, \ldots, m\}. x_i \in [l_{i,\mathcal{S}}, r_{i,\mathcal{S}}]$ (x_i = i-th element of vector x) with $l_{i,\mathcal{S}} \leq r_{i,\mathcal{S}}$. Next, we list desirable properties of preconditions.

- **High representation**. We want each specification $\mathcal{S} \in \Psi$ to capture several interesting behaviors to be meaningful and important for Ψ. With more interesting behaviors in each $\mathcal{S}$, we will need fewer specifications in Ψ, making it more interpretable. For this we define the *representation* of a specification $\mathcal{S}$ as:

$$Rep(\mathcal{S}) \triangleq \frac{|\{x \mid x \in \gamma_x \wedge \varphi_{\mathcal{S}}(x)\}|}{|\gamma_x|} \tag{1}$$

  We require each specification's precondition to contain at least $\tau_{rep} \in (0, 1]$ fraction of γ_x, i.e., $Rep(\mathcal{S}) \geq \tau_{rep}$, where τ_{rep} is a user-defined threshold.

- **High coverage**. We want all specifications in Ψ to collectively cover a large fraction of γ_x. This increases the interesting behavior information conveyed by Ψ. We define the *coverage* of Ψ (2) as the fraction of γ_x captured by any specification's precondition in Ψ.

$$Cov(\Psi) \triangleq \frac{|\bigcup_{S \in \Psi}\{x \mid x \in \gamma_x \wedge \varphi_S(x)\})|}{|\gamma_x|} \tag{2}$$

  We desire a coverage more than a user-specified threshold $\tau_{cov} \in (0, 1]$, i.e., $Cov(\Psi) \geq \tau_{cov}$.

- **Low volume**. To reduce overapproximation error due to interval-based preconditions, we want to reduce the number of inputs allowed by any specification $\mathcal{S} \in \Psi$ while maintaining high coverage and representation scores. This can be done by minimizing the volume of Ψ which is the sum of the volume of each $\varphi_{\mathcal{S}}$ (3). The parts of $\mathcal{X}$ accepted by $\varphi_{\mathcal{S}}$ are hyperrectangles denoted by the intervals $[l_{i,\mathcal{S}}, r_{i,\mathcal{S}}], \forall i \in \{1, \ldots, m\}$. Hence their volumes are defined as products of their ranges along each dimension.

$$Vol(\Psi) \triangleq \sum_{\mathcal{S} \in \Psi} \prod_{i \in \{1,\ldots,m\}} (r_{i,\mathcal{S}} - l_{i,\mathcal{S}}) \tag{3}$$

Postconditions. Let $\varphi_{\mathcal{S}}(\gamma_x) \triangleq \{x \in \gamma_x \mid \varphi_{\mathcal{S}}(x)\}$ denote the interesting behavior inputs that satisfy $\varphi_{\mathcal{S}}$. We define the postcondition $\psi_{\mathcal{S}}$ for a given precondition $\varphi_{\mathcal{S}}$ as $\psi_{\mathcal{S}}(y) \triangleq y \in \bigcup_{x \in \varphi_{\mathcal{S}}(\gamma_x)} \bigcup_{j \in [q]} \mathcal{R}_j(x)$, i.e., $\psi_{\mathcal{S}}$ accepts all the outputs of all the references for the interesting behaviors captured by $\varphi_{\mathcal{S}}$. Let $\eta_{\mathcal{S}} = |\{y \mid y \in \mathcal{Y} \wedge \psi_{\mathcal{S}}(y)\}|$ denote the number of allowed outputs by $\mathcal{S}$. Ideally, we should map each input $x \in \varphi_{\mathcal{S}}(\gamma_x)$ only to its permissible output from the references $\bigcup_{j \in [q]} \mathcal{R}_j(x)$. However, this can be difficult to satisfy with the coverage and representation constraints on precondition. Hence, we relax this requirement to overapproximate the allowed outputs of x with outputs permissible for other elements of $\varphi_{\mathcal{S}}(\gamma_x)$. To minimize the overapproximation error, we do not allow more than a specific number of outputs to be accepted by $\psi_{\mathcal{S}}$, given by a threshold $\tau_{max} \in \{1, \ldots, |\mathcal{Y} - 1|\}$, i.e., $\eta_{\mathcal{S}} \leq \tau_{max}$.

Optimization problem. For high-quality specifications, we generate specification sets Ψ satisfying a minimum coverage threshold τ_{cov}, with each specification having a representation score higher than a threshold τ_{rep} and having a maximum of τ_{max} outputs in the postcondition. With these constraints, we want to minimize the volume of Ψ. The optimal specifications set Ψ^* is the solution to the optimization problem in Equation (4). We allow the thresholds $\tau_{cov}, \tau_{rep}, \tau_{max}$ to be user-defined to generalize to varying domain-specific requirements for coverage, representation, and maximum number of outputs in postconditions.

$$\begin{aligned} &\Psi^* = argmin_{\Psi} Vol(\Psi) \\ &s.t. \quad Cov(\Psi) \geq \tau_{cov}, \quad \forall \mathcal{S} \in \Psi. Rep(\mathcal{S}) \geq \tau_{rep}, \\ &\forall \mathcal{S} \in \Psi. \eta_{\mathcal{S}} \leq \tau_{max} \end{aligned} \tag{4}$$

4 SpecTRA— Specifications from Reference Algorithms

In this section, building on our formalism from Section 3, we describe our algorithm SpecTRA, for automatically generating high-quality specifications from reference algorithms.

4.1 Practical optimization problem

The optimization problem (4) is hard to solve in general settings, as identifying all the interesting behaviors for any given references is not trivial. The references can be complex functions and may lack closed-form expression. To identify the interesting behaviors across multiple references, we need to encode them in languages such as SMT-Lib (Barrett et al., 2016) to use specialized solvers, such as Z3 (De Moura and Bjørner, 2008), which requires extensive manual efforts and may not be feasible for complicated references based on thousands of lines of intricate low-level code (e.g., for congestion control). Thus, for general applicability and ease of usage of our framework, we only assume access to the references through availability of some of their observations, from which we identify the interesting behaviors. Note that we do not assume the ability to control the observations by not assuming query-access to the references unlike prior specification generation works (Astorga et al., 2019; 2021). This causes us to use the observations available from the references in production settings, while

reducing the costs of running them in sandboxed environments. Moreover, the observations contain the behaviors of the references seen during practical deployment, which supports our quest for specifications encoding practical reliability. However, the assumption of a static, given dataset of observations imposes several constraints and SpecTRA applies the following adaptations to the optimization problem in (4) to solve it in this setting.

Interesting input *regions*. A drawback of our assuming a static, given dataset of observations is that we cannot ensure the availability of the outputs of all references for any specific input $x \in \mathcal{X}$, to identify interesting behaviors. However, we may have observations from references for 'close-by' inputs, which can indicate the behaviors of the references in a *local* input region. Let X be such an input region containing $x_1, \ldots, x_i, \ldots \in \mathcal{X}$ observations. The reference $\mathcal{R}_j$'s output $Y_{X,j} \subseteq \mathcal{Y}$ for X is the set of outputs of $\mathcal{R}_j$ for any input in X in the given observations, i.e., $Y_{X,j} \triangleq \mathcal{R}_j(X) = \bigcup_{x \in \mathcal{X}} \mathcal{R}_j(x)$. We treat a local region X as a single entity for the following discussion.

Definition 2. *(Interesting behavior regions) Interesting behavior regions $\Gamma_X \triangleq \{\ldots, X_i, \ldots\}$ is a potentially infinite set of non-overlapping local regions X_i, where each X_i contains multiple observations from each reference and $Y_{X_i} = \bigcup_{j \in [q]} Y_{X_i,j} \subset \mathcal{Y}$. Γ_y denotes the set of outputs corresponding to Γ_x, i.e., $\Gamma_y = \{\ldots, Y_{X_i}, \ldots\}$.*

The preconditions of specifications, $\varphi_{\mathcal{S}}$ are still based on intervals and an input region X is accepted by $\varphi_{\mathcal{S}}$ when all points in X satisfy the precondition $\varphi_{\mathcal{S}}(X) \iff \forall x \in X. \varphi_{\mathcal{S}}(x)$. An output y satisfies the postcondition if it is included in the output of a $X \in \Gamma_X$ for which $\varphi_{\mathcal{S}}(X)$ is true, i.e., $\psi_{\mathcal{S}}(y) \triangleq (y \in \bigcup_{X \in \Gamma_X \wedge \varphi_{\mathcal{S}}(X)} Y_X)$. Interesting behavior regions overapproximate ideal interesting behaviors introducing overapproximation errors as we map each $X \in \Gamma_X$ to outputs based on the observations in X, which may exclude some outputs for inputs of X that are not observed. We require the interesting behavior regions to be generated with more observations to reduce the error. Note that such interesting behavior regions are similar to robustness regions around given inputs as defined and used in manually designed specifications in several prior works (Chakravarthy et al., 2022; Seshia et al., 2018). However, the salient difference from the latter is that the interesting behavior regions are automatically determined using several observations of references.

Relaxed metrics. To incorporate the above notion of interesting behavior regions, Γ_X, we adapt our desirable properties of high representation (1) and high coverage (2) metrics. Originally coverage denotes the fraction of interesting behaviors captured by a specifications set Ψ. In this case, coverage modifies to the fraction of Γ_X captured in Ψ (5). We desire a coverage higher than a given threshold in the relaxed problem $\widetilde{\tau_{cov}}$, i.e., $\widetilde{Cov}(\Psi) \geq \widetilde{\tau_{cov}}$.

$$\widetilde{Cov}(\Psi) \triangleq \frac{|\bigcup_{S \in \Psi} \{X \mid X \in \Gamma_X \wedge \varphi_{\mathcal{S}}(X)\})|}{|\Gamma_X|} \tag{5}$$

Similarly, representation relaxes to the fraction of Γ_X captured in a given specification (6). We want it to be more than a given threshold $\widetilde{\tau_{rep}}$, i.e., $\forall \mathcal{S} \in \Psi. \widetilde{Rep}(\mathcal{S}) \geq \widetilde{\tau_{rep}}$.

$$\widetilde{Rep}(\mathcal{S}) \triangleq \frac{|\{X \mid X \in \Gamma_X \wedge \varphi_{\mathcal{S}}(X)\})|}{|\Gamma_X|} \tag{6}$$

The final practical optimization problem (7) thus minimizes the volume of the specifications set, while covering at least $\widetilde{\tau_{cov}}$ fraction of interesting behavior regions overall, with each specification covering at least $\widetilde{\tau_{rep}}$ fraction of interesting behavior regions and allowing less than τ_{max} outputs.

$$\begin{aligned} &\widetilde{\Psi^*} = argmin_{\Psi} Vol(\Psi) \\ &s.t. \quad \widetilde{Cov}(\Psi) \geq \widetilde{\tau_{cov}}, \quad \forall \mathcal{S} \in \Psi. \widetilde{Rep}(\mathcal{S}) \geq \widetilde{\tau_{rep}}, \\ &\forall \mathcal{S} \in \Psi. \eta_{\mathcal{S}} \leq \tau_{max} \end{aligned} \tag{7}$$

4.2 Implementation

Figure 1 presents a high-level overview of SpecTRA's working. Algorithm 1 gives the pseudocode of SpecTRA's algorithm to solve (7). Let $\mathcal{D}_1, \ldots \mathcal{D}_q$ be the set of observations from the references $\mathcal{R}_1, \ldots, \mathcal{R}_q$ respectively.

Identifying interesting behavior regions. To identify the interesting behavior regions across references, Algorithm 1 first identifies non-overlapping local input regions, which could potentially be interesting, from $\mathcal{X}$. We form the input regions by partitioning $\mathcal{X}$ along each dimension into a fixed number $p > 0$ of equally-sized parts (line 4). Input regions may be identified with other partitioning methods too, however, SpecTRA is agnostic to them. We selected this partitioning method for simplicity. Not all input regions thus obtained can be worth considering, as some may have very few or 0 observations of some references. Therefore, we identify the important input regions having at least a certain fraction of the available observations for every reference (line 5) to reduce the overapproximation error due to considering input regions. We then obtain the interesting behavior regions from the important regions common across all the references by applying the condition for interesting behaviors (Definition 1). We thus obtain the set of input regions Γ_X and their corresponding outputs Γ_y from $\mathcal{D}_1, \ldots, \mathcal{D}_q$ as interesting behavior regions (line 6).

Combining interesting behavior regions into specifications. The identified interesting behavior regions are used

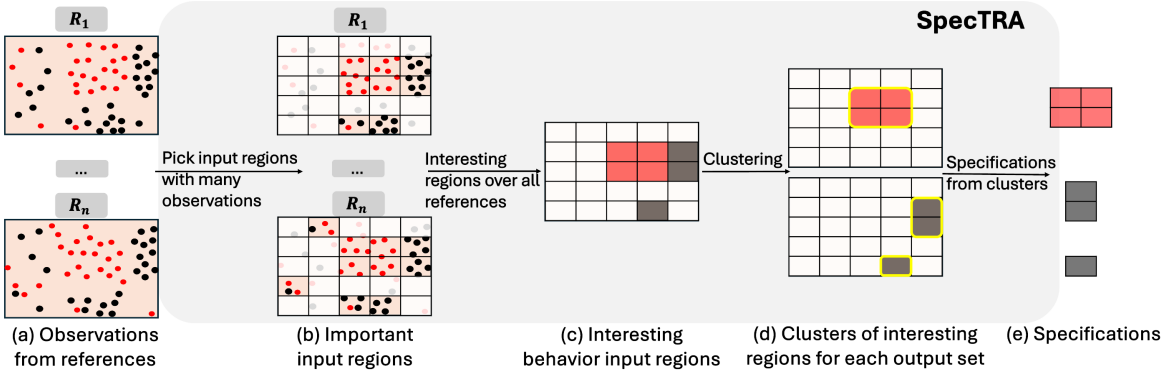


Figure 1: (Overview of SpecTRA) Observations from all the references (a) are given as input to our specification generation algorithm, SpecTRA. For illustration purposes, a 2-D feature space is considered and each input is classified in either the red or black class by the corresponding reference algorithm. SpecTRA takes the observations and from them develops a set of likely specifications (e). To generate the specifications, SpecTRA first partitions the input space into input regions and from them identifies the regions which have several observations for each reference separately (b) to form the important input regions for the reference. These important regions are combined into interesting behavior input regions (c) according to Definition 1. Clusters are identified in the interesting behavior regions with the same output set according to density (d). These clusters form the specifications (e) that map the inputs in each cluster to the corresponding output set of the cluster.

Algorithm 1 SpecTRA Algorithm

1: **Input:** $\mathcal{X}, \mathcal{Y}, \mathcal{D}_1, \ldots \mathcal{D}_q, \widetilde{\tau_{cov}}, \widetilde{\tau_{rep}}, \tau_{max}, p$
2: **Output:** Ψ
3: $\Psi \leftarrow \emptyset$
{ Identifying interesting behavior regions}
4: input_regions $\leftarrow$ `Partition`$(\mathcal{X}, p)$
5: $\mathcal{I} \leftarrow$ `Important`(input_regions, $\mathcal{D}_1, \ldots, \mathcal{D}_q$)
6: $\Gamma_X, \Gamma_y \leftarrow$ `Interesting`$(\mathcal{I})$
{Combining interesting behavior regions}
7: **for** $i \in [1, \ldots, \tau_{max}]$ **do**
8: **for** each subset $\Omega \subset \mathcal{Y}$ with $|\Omega| = i$ **do**
9: $\Gamma_{\Omega,x} \leftarrow \{X \mid X \in \Gamma_X, Y_X \subseteq \Omega\}$
10: $\Gamma_{\Omega,y} \leftarrow \{Y_X \mid X \in \Gamma_X\}$
11: $\mathcal{L} \leftarrow$ `Cluster`$(\Gamma_{\Omega,x}, \widetilde{\tau_{rep}})$
12: $\mathcal{S}_1, \ldots, \mathcal{S}_l \leftarrow$ `Cluster2Spec`$(\mathcal{L}, \Gamma_{\Omega,x}, \Gamma_{\Omega,y})$
13: $\Psi.extend(\mathcal{S}_1, \ldots, \mathcal{S}_l)$
14: **if** $\widetilde{Cov}(\Psi) \geq \widetilde{\tau_{cov}}$ **then**
15: **return** Ψ
16: **return** Ψ

to solve the optimization problem in (7). Firstly, SpecTRA considers the constraint on the maximum number of outputs accepted by the postconditions, i.e., $\forall \mathcal{S} \in \Psi. \eta_{\mathcal{S}} \leq \tau_{max}$. To satisfy the constraint, SpecTRA forms individual specifications only over X $(\in \Gamma_X)$ which combine to yield a postcondition accepting less than τ_{max} outputs. To do so, SpecTRA enumerates all subsets Ω of $\mathcal{Y}$, such that $|\Omega| \in [1, \ldots, \tau_{max}]$. For each Ω, it filters out the interesting behavior regions having output as a subset of Ω (lines 9-10). The problem thus reduces to generating specifications for $\Gamma_{\Omega,x}, \Gamma_{\Omega,y}$ without any constraints on the postconditions.

The precondition of each specification that allows any subset of $\Gamma_{\Omega,x}$ should be the tightest bounding hypercube the subset, to minimize the volume of the specification. Moreover, as all input regions have the same volume, minimizing the volume of specification sets reduces to minimizing the number of input regions allowed by the preconditions of the specifications, while satisfying the representation and coverage constraints. The optimization problem becomes minimizing the number of input regions in the preconditions (volume) while retaining a minimum number of interesting behavior input regions in every specification (representation) and over all the specifications (coverage). Exactly solving this optimization problem is hard, as (i) the search space is large with several observations, high number of parts p along each input dimension, high number of input dimensions, (ii) there are no obvious structures (e.g., decomposability or differentiability) that we can exploit in our setting where we cannot query or access the reference implementations. Hence, we develop an approximate solution by identifying clusters in $\Gamma_{\Omega,x}$ according to the proximity of constituent regions. To minimize the volume, we apply density-based clustering methods such as the DBSCAN algorithm (Ester et al., 1996) as they find dense clusters and are resistant to outliers. The general `Cluster`(.) function in line 11 uses the selected clustering algorithm to returns l clusters from $\Gamma_{\Omega,x}$, denoted as cluster labels $\mathcal{L} \in [1, \ldots, l]^{|\Gamma_{\Omega,x}|}$ for each element in $\Gamma_{\Omega,x}$. We encode the representation constraint into the clustering process, requiring each cluster to have at least $\widetilde{\tau_{rep}}$ samples. Each cluster thus identified can be combined to form a specification (line 12). Specif-

ically, each $\mathcal{S}_i, i \in [1, \ldots, l]$ has $\varphi_{\mathcal{S}_i}$ as the tightest hyper-rectangle bounding $\mathcal{R} \triangleq \{X \mid X \in \Gamma_{\Omega,x}, \mathcal{L}(X) = i\}$, where $\mathcal{L}(X)$ denotes the cluster label for input region X. $\psi_{\mathcal{S}_i} = \bigcup_{X \in \mathcal{R}} Y_X$, i.e., the post condition allows for the outputs corresponding to any element of $\Gamma_{\Omega,x}$ in the i^{th} cluster. For the coverage constraint, we adopt a best-effort approach (lines 14-15), wherein we keep generating specifications till either $\widetilde{\tau_{cov}}$ is achieved or all options for Ω are exhausted.

5 EXPERIMENTS

In this section, we study the *quality* and *utility* of SpecTRA's specifications. We illustrate SpecTRA's specifications in two applications having reference algorithms and their neural counterparts — Adaptive Bit Rate (ABR) algorithms in video streaming and Congestion Control (CC) algorithms. Pensieve (Mao et al., 2017) is a popular neural network (NN) based RL-agent for ABR that decides the bit rate for the next video chunk. Pensieve needs to choose from 6 possible bit rates for the next video chunk — $300, 750, 1200, 1850, 2850, 4300$ kbps. While ABR has several traditional (reference) algorithms, there are two salient ones, that we use as references for generating specifications for Pensieve — the Buffer-based (BB) algorithm (Huang et al., 2014) and the Model Predictive Control (MPC) algorithm (Yin et al., 2015). Aurora (Jay et al., 2019) is a popular NN-based RL-agent for CC that proposes real-valued changes to the rate of sending packets over a network to reduce congestion. The CC references that we consider for generating specifications for Aurora are the BBR (Cardwell et al., 2017) and Cubic (Ha et al., 2008) algorithms. Note that our framework is general to handle more than two references. However, we expect to get fewer interesting behavior regions with additional references, which may deteriorate the quality of the specifications. Hence, we choose to limit to two references for each application. As SpecTRA uses several thresholds and parameters to generate specifications, we show an ablation study (Appendix D) to know their effects on the specifications' quality and generate specifications with the best settings.

5.1 Experimental setup

We conducted our experiments on a 12th Gen 20-core Intel i9 processor. We collect the observations from references in the training environments of the NN, to generate relevant specifications. For ABR, as we have the implementations of the references readily available, we run them in the training environment of the target model Pensieve, with its training traces that govern the network characteristics for video streaming at each time step. We use the publicly available training dataset of Pensieve consisting of 128 traces, each having 100s of time steps. We test the specifications on Pensieve's test set that consists of 143 traces, also having 100s of time steps each. For CC, however, the reference algorithms are embedded in the operating system kernels, making them less amenable to run in the target model Aurora's training environment. Hence, we obtain observations for them from their execution logs in the Pantheon project (Yan et al., 2018). We retrain the Aurora models using some of the corresponding network traces from Pantheon so as to align the reference observations with the training environment of the models. We use 75% traces for training and remaining for testing the Aurora models. We describe the details of the retraining and mention the specific traces used in training and testing in Appendix B. We experiment with both the retrained and original Aurora models. We generate specifications for both applications with the observations from references on the training traces and use the observations from the testing traces to evaluate the specifications.

As SpecTRA assumes a finite discrete set of outputs, which is not the case for Aurora (output is real-valued change of packet sending rate), we discretize the output using the sign function, which gives the sign of the change resulting in 3 possible outcomes: '+', '-', and '0'. SpecTRA uses the DBSCAN (Ester et al., 1996) clustering algorithm to solve the optimization problem in (7). We detail the settings of DBSCAN in Appendix C. SpecTRA develops the specifications on a subset of input features of the target NN for which we empirically observe the best quality of specifications. For ABR, SpecTRA uses the current buffer size and the download times observed in the last 3 time steps. For CC, SpecTRA uses the history of the latency gradient, latency ratio, and sending ratio features used by Aurora, over previous 4 time steps. We have selected these specific features for the two applications following those in the manually-designed specifications in Eliyahu et al. (2021) and selected the history of the features in specifications with an ablation study in Appendix D. We keep the coverage threshold, τ_{cov} to be 1, so as to get specification sets with the highest possible coverage. SpecTRA generates specifications for either application in less than 30 seconds.

5.2 Quality of Specifications

5.2.1 Quantitative analysis

To evaluate the quality of the specifications, we check them against the observations from the references over the training and testing environments for the neural models. We use the specification evaluation metrics of *support* and *confidence* inspired from prior specification mining work, Lemieux et al. (2015) and data mining literature (Han et al., 2011). We formally define these metrics next.

We want the specifications to cover most of the observations $\mathcal{D}_j$ from each reference $\mathcal{R}_j$ to correctly describe their behavior. For this, we measure the fraction of observations

Application	Reference	Observation type	Support (SpecTRA)	Confidence (SpecTRA)	Support (prior)	Confidence (prior)
ABR	BB	Training	0.95	1.0	0.03	1.0
		Test	0.96	1.0	0.01	0.99
	MPC	Training	0.64	0.87	0.09	0.87
		Test	0.59	0.87	0.07	0.84
	Pensieve (small)	Training	0.17	0.99	0.27	0.86
		Test	0.13	0.99	0.25	0.85
	Pensieve (mid)	Training	0.31	0.96	0.14	0.93
		Test	0.26	0.97	0.13	0.91
	Pensieve (big)	Training	0.26	0.97	0.22	0.93
		Test	0.21	0.98	0.18	0.92
CC	BBR	Training	0.89	0.75	0.01	0.43
		Test	0.74	0.81	0.01	0.38
	Cubic	Training	0.97	0.82	0.001	0.29
		Test	0.79	0.79	0.07	0.42
	Aurora (small)	Training	1.0	1.0	0.01	1.0
		Test	1.0	1.0	0.01	1.0
	Aurora (mid)	Training	0.98	1.0	0.17	0.03
		Test	0.96	1.0	0.34	0.01
	Original Aurora	Training	1.0	1.0	0.01	1.0
		Test	0.99	1.0	0.05	0.05

Table 1: Support and confidence of SpecTRA's specifications and that of the specifications in (Eliyahu et al., 2021) on the training and testing observations of the references and NNs for both the ABR and Congestion Control (CC) applications

from $\mathcal{D}_j$ that are accepted by the precondition $\varphi_{\mathcal{S}}$ of any specification $\mathcal{S}$ in SpecTRA's generated specifications set Ψ. This quantity is the support $Sup(\Psi, \mathcal{R}_j)$ (8) of Ψ for $\mathcal{R}_j$.

$$Sup(\Psi, \mathcal{R}_j) \triangleq \frac{|\{x|(x, \mathcal{R}_j(x)) \in \mathcal{D}_j \wedge \bigcup_{\mathcal{S}\in\Psi} \varphi_{\mathcal{S}}(x)\}|}{|\mathcal{D}_j|} \quad (8)$$

Let $\mathcal{D}_{j,\Psi} \triangleq \{x|(x, \mathcal{R}_j(x)) \in \mathcal{D}_j \wedge \bigcup_{\mathcal{S}\in\Psi} \varphi_{\mathcal{S}}(x)\}$ denote the observations from reference $\mathcal{R}_j$ contributing to the support for Ψ. Alongside support, we want the specifications to be correct on the observations. Thus, we check the fraction of instances in $\mathcal{D}_{j,\Psi}$ where $\forall \mathcal{S} \in \Psi. \varphi_{\mathcal{S}} \implies \psi_{\mathcal{S}}$, i.e., postconditions of all specifications are satisfied whose preconditions hold for observations contributing to the support of Ψ. We call this the confidence $Conf(\Psi, \mathcal{R}_j)$ for Ψ (9).

$$Conf(\Psi, \mathcal{R}_j) \quad (9)$$
$$\triangleq \frac{|\{x|x \in \mathcal{D}_{j,\Psi} \wedge \forall \mathcal{S} \in \Psi. (\varphi_{\mathcal{S}}(x) \implies \psi_{\mathcal{S}}(\mathcal{R}_j(x)))\}|}{|\mathcal{D}_{j,\Psi}|}$$

We report the support and confidence for SpecTRA's specifications for both applications in Table 1. We compare our specifications with those given by prior works (Brix et al., 2023; Eliyahu et al., 2021). We find that the specifications in VNN-COMP 2023 (Brix et al., 2023) have 0 support over the observations for the references in both applications. Hence, we do not include them in our study. The specifications in (Eliyahu et al., 2021) are temporal in nature and, therefore, not directly comparable. Hence, we use the negation of their specifications for bad system states for individual transitions and compare with our specifications. We present the support and confidence of the prior specifications in Eliyahu et al. (2021) in Table 1. The total number of training and testing observations for the ABR algorithms are 77981 and 27837 respectively. We filter out the observations for CC to consist of those that fall within the observation space on which the Aurora models are typically trained. The total number of training and testing observations for the CC references are $\sim 130k$ and $\sim 40k$ respectively, and those for the Aurora models are $\sim 140k$ and $\sim 200k$ respectively.

The specifications in the prior work have low support for the observations from the references and similar confidence as our specifications. These results indicate that our specifications correctly encode more of the trusted behaviors of the references over the relevant (training and testing) input distributions of the NNs than the existing specifications. The existing specifications show comparable support over the training and testing observations from Pensieve models but lower confidence than our specifications.

5.2.2 *Qualitative analysis with case studies*

ABR. SpecTRA generates a specification set with 30 specifications for Pensieve models. We present some specifications from the generated set next, to demonstrate their quality and conformance with intuitively correct behavior. Note that all specifications are in conjunction, so they need to hold simultaneously for the satisfaction of the generated specifications set. We give the entire specification set for Pensieve in Ap-

pendix E. Note that the ranges of buffer size (BS — duration of pre-retrieved video stored in the buffer) and video chunk download-time (DT) features are $[4, 60]$ seconds and $(0, \infty)$ respectively. Each video chunk is 4 seconds long.

Specification 1a shows the conditions for which the lowest 2 bitrates — 300 and 750 kbps are allowed by the postcondition. These conditions consist of low BS and > 2 seconds of DTs. Intuitively, the ABR algorithm should output low bit rates for such states of video streaming systems, as the buffer does not have enough video chunks to render and there is some delay in downloading new video chunks. In such scenarios, to prevent rebuffering, the video chunks should be fetched at lower bit rates. Manually-designed specifications, such as those in Eliyahu et al. (2021) capture only extreme behaviors, such as situations when the lowest bit rate must be output by the ABR algorithm. However, SpecTRA's specifications can encode intermediate behaviors, such as cases when the lowest 2 bit rates can be permissible, as well. Specification 1b shows cases where we specify that the ABR algorithm does not output the lowest bit rate. We specify that the buffer should contain more than 2 video chunks, each of which is 4 seconds long, and the DT should be a moderate value for the lowest bit rate to not get selected. Note that, this specification supplements the intuitive specification about avoiding the lowest bit rate in Eliyahu et al. (2021). The prior work's specification forbids the lowest bit rate when BS is > 4 seconds and the DTs are < 4 seconds, whereas SpecTRA's specification disallows the lowest bit rate even when DTs can be > 4 seconds, with large enough buffer. The prior work does not specify the desirable behavior for > 4 seconds of DT with BS > 4 seconds. Moreover, the permissible ranges of the input features at different points in their history can vary in the preconditions of automatically generated specifications, unlike those in manually-designed, intuition-based specifications. Obtaining such fine-grained specifications is beyond the scope of manually-designed specifications but can be achieved using automated methods such as SpecTRA.

Precondition	**Precondition**
$BS \in [4.0, 5.0]$,	$BS \in [10.9, 12.3]$,
$DT[-1] \in [2.8, 6.6]$,	$DT[-1] \in [4.1, 7.9]$,
$DT[-2] \in [2.8, 6.6]$,	$DT[-2] \in [1.5, 6.6]$,
$DT[-3] \in [5.4, 9.2]$	$DT[-3] \in [1.5, 5.4]$
Postcondition	**Postcondition**
$BR \in \{300, 750\}$	$BR \in \{750, 1200, 1850, 2850, 4300\}$
(a)	(b)

Specifications 1: Conjunctive specifications for ABR. (BS: Buffer Size, DT$[-i]$: i^{th} last download time, BR: Bit Rate)

Precondition
$LG[-1] \in [-1.0, 0.29], LG[-2] \in [-0.78, 0.07]$,
$LG[-3] \in [-0.78, 0.29], LG[-4] \in [-1.0, 0.29]$
$LR[-1] \in [1.0, 1.88], LR[-2] \in [1.0, 1.88]$,
$LR[-3] \in [1.0, 1.88], LR[-4] \in [1.0, 1.88]$
$SR[-1] \in [0.0, 17.18], SR[-2] \in [0.0, 17.18]$,
$SR[-3] \in [0.0, 17.18], SR[-4] \in [0.0, 17.18]$
Postcondition
`Change in Sending Rate` $\in \{+, -\}$

Specifications 2: For CC. (LG: latency gradient, LR: latency ratio, SR: sending ratio, x$[-i]$: i^{th} last value of x)

CC. SpecTRA generates 1 specification, shown in Specification 2, for Aurora models. Interestingly, this specification's precondition contains the precondition of the specification allowing non-zero change of sending rate in its postcondition in Eliyahu et al. (2021). The latter specification comprises of a very small fraction of SpecTRA's specification. For example, all latency gradients are specified to be within $[-0.01, 0.01]$, latency ratios in $[1.0, 1.01]$, and sending ratios as only 1.0. Thus, the corresponding prior specification is conservative, probably due to its manual design.

5.3 Utility of Specifications

5.3.1 NN Verification

Next, we explore a popular downstream application of NN specifications — verifying trained NNs. We attempt to verify the NNs in each application for SpecTRA's specifications using the SOTA complete-verifier, $\alpha\beta$-CROWN (Xu et al., 2020). As the overall specification set is a conjunction of all elements in the specification set generated by SpecTRA, we attempt to verify each specification in the set individually by encoding it in the VNN-Lib format (Demarchi et al., 2023). SpecTRA generates a specification set containing 30 specifications for ABR and 1 specification for the CC setting. We set $\alpha\beta$-CROWN's timeout as 10 minutes. We use the more precise activation splitting for the Pensieve models and input-splitting for Aurora models. This is because $\alpha\beta$-CROWN does not support activation splitting for regression models currently, to the best of our knowledge. As our specifications encode only a subset of the inputs in the preconditions, we specify the other input features as their ranges as seen in the observations used to generate the specifications. Table 2 presents our findings for both applications. The *Verified* instances occur when the specification is satisfied by the model, the *Falsified* instances are when we can find a successful attack for the specification on the model, and *Timeout* is when the verifier times out. We find that none of the Pensieve models satisfy the overall conjunctive

Model	Verified	Falsified	Timeout
Pensieve (small)	5	20	5
Pensieve (mid)	2	22	6
Pensieve (big)	1	28	1
Aurora (small)	0	0	1
Aurora (mid)	0	0	1
Original Aurora	0	0	1

Table 2: Verifying SpecTRA's specifications sets consisting of 30 specifications for ABR and 1 specification for CC.

specification, as some of the constituent specifications can be falsified. This indicates that the models, while optimizing for average reward, may not be trustworthy for practical usage. Moreover, we see that the SOTA verifier times out for some of the specifications for both Pensieve and Aurora. This suggests that the specifications are challenging for contemporary verifiers and can be used to guide the design of customized verifiers for NNs in computer systems.

5.3.2 Targeted attacks on NNs

Next, we attack Pensieve to falsify SpecTRA's specification and study the attacks qualitatively. We generate Projected Gradient Descent attacks (Madry et al., 2019) on the models to identify inputs that cause the models to give extreme outputs (lowest/highest bit rates). As SpecTRA's overall specifications set is a conjunction of specifications, violating a single specification will falsify the specifications set. Hence, we attack the specifications from the generated specifications set that does not allow the extreme outputs in their postconditions. We show only the features of the attack input specified by the specifications. These features are sufficient to show the violation of intuitive behavior from the models. We show attacks on the Pensieve (big) model and note that similar attacks exist for other models too.

Input :	**Input** :
$BS = 11.2,$	$BS = 4.0,$
$DT[-1] = 6.9,$	$DT[-1] = 11.8,$
$DT[-2] = 2.9,$	$DT[-2] = 0.2,$
$DT[-3] = 2.0$	$DT[-3] = 0.2$
Output : $BR = 300$	**Output** : $BR = 4300$
(a)	(b)

Attack (a) consists of an input where the BS is high and only the last DT is high, with the other DTs low. The model still conservatively predicts the lowest bit rate, while a higher bit rate could be supported by the system. The Buffer-based (BB) algorithm, a simple ABR algorithm, can also predict a higher bitrate (1850 kbps) for this case. Attack (b), on the other hand, consists of an input where the buffer consists of only 1 video chunk and the previous DT had been high. For this input, the model predicts the highest bit rate, which may result in rebuffering of the system and, therefore, affect the quality of experience for the users. The simple BB reference algorithm predicts 300 kbps for this instance.

6 Related Work

Specifications for neural networks. The current approach to generate specifications for neural networks (NNs) is largely dependent on human design. Many existing works, such as (Eliyahu et al., 2021; Wu et al., 2022; Wei et al., 2023), rely on experts to design their specifications. Also, in the International Verification of Neural Networks Competition, VNN-Comp (Brix et al., 2023), expert-designed specifications are used in benchmarks, including for Adaptive Bit Rate and Congestion Control. However, the quality and relevance of these expert-designed specifications remain unclear. Recent work (Geng et al., 2023) proposes to automatically mine neural activation patterns (NAP) as specifications. NAP refers to the pattern of activation functions—whether they are activated or deactivated—given a specific neural network and an input. Geng et al. (2024) introduces multiple approaches to mine NAPs for a given neural network. SpecTRA differs from NAP mining as SpecTRA's specifications can generalize beyond the target neural networks and can be intuitively validated with domain knowledge.

Specification generation for programs. There is a long history of work focused on synthesizing specifications for traditional programs, which has inspired SpecTRA. Unlike prior work (Ernst et al., 1999; Ammons et al., 2002; Park et al., 2023; Astorga et al., 2019; 2021), SpecTRA targets neural networks instead of traditional general programs. To the best of our knowledge, SpecTRA is the first to mine specifications for neural networks in computer systems using reference algorithms. Astorga et al. (2023) also synthesize contracts for neural networks, but they operate in a setting with query-access to an oracle, which is not practically extensible to the applications we study.

NN verification. NN verification formally verifies given neural networks for desirable properties such as robustness to input perturbations. It can be broadly classified as complete (Jaeckle et al., 2021; Ferrari et al., 2022; Xu et al., 2021) and incomplete (Xu et al., 2020; Singh et al., 2019) verification. NN verification is NP-complete (Katz et al., 2017), which is hard to scale to larger NNs. However, the NNs in computer systems are generally small due to efficiency requirements and hence are conducive to verification.

7 Conclusion

We present an automated approach for generating specifications for neural networks in applications where trustworthy reference algorithms exist. We formalize specification generation as an optimization problem and propose an effective algorithm SpecTRA. We show specifications for two important applications — adaptive bit rate setting and congestion control. We analyze the quality of SpecTRA's specifications and use them to verify and identify previously unknown vulnerabilities in SOTA neural networks.

References

Aws Albarghouthi, Isil Dillig, and Arie Gurfinkel. Maximal specification synthesis. *SIGPLAN Not.*, 51(1): 789–801, January 2016. ISSN 0362-1340. doi: 10.1145/2914770.2837628. URL https://doi.org/10.1145/2914770.2837628.

Glenn Ammons, Rastislav Bodík, and James R. Larus. Mining specifications. *SIGPLAN Not.*, 37(1):4–16, January 2002. ISSN 0362-1340. doi: 10.1145/565816.503275. URL https://doi.org/10.1145/565816.503275.

Angello Astorga, P. Madhusudan, Shambwaditya Saha, Shiyu Wang, and Tao Xie. Learning stateful preconditions modulo a test generator. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 775–787, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367127. doi: 10.1145/3314221.3314641. URL https://doi.org/10.1145/3314221.3314641.

Angello Astorga, Shambwaditya Saha, Ahmad Dinkins, Felicia Wang, P. Madhusudan, and Tao Xie. Synthesizing contracts correct modulo a test generator. *Proc. ACM Program. Lang.*, 5(OOPSLA), oct 2021. doi: 10.1145/3485481. URL https://doi.org/10.1145/3485481.

Angello Astorga, Chiao Hsieh, P. Madhusudan, and Sayan Mitra. Perception contracts for safety of ml-enabled systems. *Proc. ACM Program. Lang.*, 7(OOPSLA2), October 2023. doi: 10.1145/3622875. URL https://doi.org/10.1145/3622875.

Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples, 2018. URL https://arxiv.org/abs/1707.07397.

Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. *SIGCOMM Comput. Commun. Rev.*, 43(4):339–350, aug 2013. ISSN 0146-4833. doi: 10.1145/2534169.2486025. URL https://doi.org/10.1145/2534169.2486025.

Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. Certifying geometric robustness of neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/

f7fa6aca028e7ff4ef62d75ed025fe76-Paper.pdf.

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

Osbert Bastani, Saswat Anand, and Alex Aiken. Specification inference using context-free language reachability. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, page 553–566, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333009. doi: 10.1145/2676726.2676977. URL https://doi.org/10.1145/2676726.2676977.

Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results, 2023.

Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: congestion-based congestion control. *Commun. ACM*, 60(2):58–66, jan 2017. ISSN 0001-0782. doi: 10.1145/3009824. URL https://doi.org/10.1145/3009824.

Arnav Chakravarthy, Nina Narodytska, Asmitha Rathis, Marius Vilcu, and Gagandeep Singh. Property-driven evaluation of rl-controllers in self-driving datacenters. 2022. URL https://api.semanticscholar.org/CorpusID:254568173.

Isha Chaudhary, Alex Renda, Charith Mendis, and Gagandeep Singh. Comet: Neural cost model explanation framework. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 499–511, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/eb261df4322a8bd0a73093c4d8a0d02d-Paper-Conference.pdf.

Leonardo De Moura and Nikolaj Bjørner. 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*, page 337–340, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3540787992.

Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. Supporting standardization of neural networks verification with vnnlib and coconet. 10 2023. doi: 10.29007/5pdh.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-Learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, Renton, WA, April 2018. USENIX Association. ISBN 978-1-939133-01-4. URL https://www.usenix.org/conference/nsdi18/presentation/dong.

Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021.

Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, page 213–224, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130740. doi: 10.1145/302405.302467. URL https://doi.org/10.1145/302405.302467.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models, 2018. URL https://arxiv.org/abs/1707.08945.

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2022.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018. doi: 10.1109/SP.2018.00058.

Chuqin Geng, Nham Le, Xiaojie Xu, Zhaoyue Wang, Arie Gurfinkel, and Xujie Si. Towards reliable neural specifications. In *International Conference on Machine Learning*, pages 11196–11212. PMLR, 2023.

Chuqin Geng, Zhaoyue Wang, Haolin Ye, Saifei Liao, and Xujie Si. Learning minimal nap specifications for neural network verification. *arXiv preprint arXiv:2404.04662*, 2024.

Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011. ISBN 0123814790.

Haoyu He, Tianhao Wei, Huan Zhang, Changliu Liu, and Cheng Tan. Characterizing neural network verification for systems with NN4SysBench. In *Workshop on Formal Verification of Machine Learning*. ICML, 2022.

C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, oct 1969.

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 *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

Florian Jaeckle, Jingyue Lu, and M. Pawan Kumar. Neural network branch-and-bound for neural network verification, 2021. URL https://arxiv.org/abs/2107.12855.

Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

Huiling Jiang, Qing Li, Yong Jiang, Gengbiao Shen, Richard Sinnott, Chen Tian, and Mingwei Xu. When machine learning meets congestion control: A survey and comparison, 2020. URL https://arxiv.org/abs/2010.11397.

Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks, 2017. URL https://arxiv.org/abs/1702.01135.

Tim Kraska. Towards instance-optimized data systems. *Proceedings of the VLDB Endowment*, 14(12), 2021.

Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General ltl specification mining (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 81–92, 2015. doi: 10.1109/ASE.2015.71.

Zikun Liu, Changming Xu, Emerson Sie, Gagandeep Singh, and Deepak Vasisht. Exploring practical vulnerabilities of machine learning-based wireless systems. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1801–1817, Boston, MA, April 2023. USENIX Association. ISBN 978-1-939133-33-5. URL https://www.usenix.org/conference/nsdi23/presentation/liu-zikun.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019. URL https://arxiv.org/abs/1706.06083.

Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, 2017.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters, 2019. URL https://arxiv.org/abs/1810.01963.

Charith Mendis, Alex Renda, Saman Amarasinghe, and Michael Carbin. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks, 2019. URL https://arxiv.org/abs/1808.07412.

Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting deep learning-based networking systems. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 154–171, 2020.

Matthew Mirman, Timon Gehr, and Martin Vechev. Robustness certification of generative models, 2020. URL https://arxiv.org/abs/2004.14756.

Kanghee Park, Loris D'Antoni, and Thomas Reps. Synthesizing specifications, 2023.

Yusuf Sani, Andreas Mauthe, and Christopher Edwards. Adaptive bitrate selection: A survey. *IEEE Communications Surveys & Tutorials*, 19:2985–3014, 2017. URL https://api.semanticscholar.org/CorpusID:10803927.

Sanjit A. Seshia, Ankush Desai, Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte, and Xiangyu Yue.

Formal specification for deep neural networks. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis*, pages 20–34, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01090-4.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), January 2019. doi: 10.1145/3290354. URL https://doi.org/10.1145/3290354.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014. URL https://arxiv.org/abs/1312.6199.

Tianhao Wei, Zhihao Jia, Changliu Liu, and Cheng Tan. Building verified neural networks for computer systems with ouroboros. *Proceedings of Machine Learning and Systems*, 5, 2023.

Cheng Wen, Jialun Cao, Jie Su, Zhiwu Xu, Shengchao Qin, Mengda He, Haokun Li, Shing-Chi Cheung, and Cong Tian. Enchanting program specification synthesis by large language models using static analysis and program verification, 2024. URL https://arxiv.org/abs/2404.00762.

Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 459–471, Lombard, IL, April 2013. USENIX Association. ISBN 978-1-931971-00-3. URL https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/winstein.

Haoze Wu, Clark Barrett, Mahmood Sharif, Nina Narodytska, and Gagandeep Singh. Scalable verification of gnn-based job schedulers. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):1036–1065, 2022.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33, 2020.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, and Bhavya Kailkhura. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2021.

Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, Boston, MA, July 2018. USENIX Association. ISBN 978-1-939133-01-4. URL https://www.usenix.org/conference/atc18/presentation/yan-francis.

Rem Yang, Jacob Laurel, Sasa Misailovic, and Gagandeep Singh. Provable defense against geometric transformations. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. *SIGCOMM Comput. Commun. Rev.*, 2015.

G.P. Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000. doi: 10.1109/5326.897072.

A PENSIEVE'S ARCHITECTURAL DETAILS

Pensieve's original architecture (Mao et al., 2017), that corresponds to our mid model has the following structure:

First Layer: 3 parallel fully connected layer, each contains 128 neurons, and an 1D convolution layer with 128 filters and kernel size is 4. These 4 layers take the input features in parallel.
Second Layer: A fully connected (linear) layer with 128 neurons.
Output Layer: A fully connected (linear) layer with 6 neurons.

Following the NN4Sys benchmarks in VNN Comp 2024 (https://sites.google.com/view/vnn2024), we also include the small and big models for Pensieve, having the following architectures.

Small
First Layer: 4 parallel fully connected layer, each contains 128 neurons.
Second Layer: A fully connected (linear) layer with 128 neurons.
Output Layer: fully connected (linear) layer with 6 neurons.

Big
First Layer: 3 parallel fully connected layer, each contains 128 neurons, and an 1D convolution layer with 128 filters and kernel size is 4. These 4 layers are parallel.
Second Layer: A fully connected (linear) layer with 256 neurons.
Output Layer: fully connected (linear) layer with 6 neurons.

B TRAINING AURORA

Aurora model architectures. In this paper, we provide two different architectures for the Aurora model: the *small model* and the *mid model*. The *mid model* retains the same architecture as the initial Aurora policy agent from the original paper (Jay et al., 2019), which utilized a fully-connected neural network with two hidden layers of $32 \rightarrow 16$ neurons and employed a *tanh* nonlinearity function. We have also developed a *small model* with a similar architecture but scaled down to two hidden layers of $16 \rightarrow 8$ neurons, also using the *tanh* nonlinearity.

Training and testing setting. The original Aurora model was trained in a gym simulation environment designed to replicate network links, with bandwidth and latency randomized to reflect real-world conditions.

To adapt Aurora for conditions similar to those experienced by BBR and Cubic, we made slight modifications to the simulation. This included using varied bandwidth from Pantheon traces (11 in total, available at Pantheon Traces), as well as adjusting loss rate, packet queue size, and one-way delay. To ensure broad coverage of bandwidth scenarios, we increased the training steps.

In training, we simulate network conditions using Pantheon traces. Each condition includes:

- **Bandwidth Trace**: Patterns of bandwidth over time.
- **Loss Rate**: Percentage of packets dropped.
- **Delay**: Time for a packet to travel one way.
- **Queue Size**: Maximum packets held in the network buffer before forwarding or dropping.

Using 11 traces, we had 18 distinct network conditions by varying loss, delay, and queue size. We split these conditions, with 75% used for training and 25% for testing, ensuring the RL-based Aurora model is not exposed to test patterns during training. During training and testing, we ensure that all testing network conditions are run at least once.

In our experiment, with a fixed random seed of 0, the split is as follows:

- **Training Conditions**: 13, 14, 2, 5, 9, 8, 7, 15, 18, 6, 4, 11, 16
- **Testing Conditions**: 1, 3, 10, 12, 17

Network condition details can refer below:

No.	Trace File	Delay	Loss	Queue Size
1	0.57mbps-poisson	28	0.0477	14
2	2.64mbps-poisson	88	0	130
3	3.04mbps-poisson	130	0	426
4	100.42mbps	27	0	173
5	77.72mbps	51	0	94
6	114.68mbps	45	0	450
7	12mbps	10	0	1
8	60mbps	10	0	1
9	108mbps	10	0	1
10	12mbps	50	0	1
11	60mbps	50	0	1
12	108mbps	50	0	1
13	0.12mbps	10	0	10000
14	10-every-200	10	0	10000
15	12mbps	30	0	6
16	12mbps	30	0	20
17	12mbps	30	0	30
18	12mbps	30	0	60

Table 3: Mapping of Network Parameters to Trace Files

More details can be found in our network simulation implementation.

Aurora Model Performance. We evaluate the Aurora model on our testing network conditions. To benchmark its performance, we include a random model, which is randomly initialized and untrained. The *Original* model refers to models trained in Aurora's original RL environment, while *Current* refers to the model we trained under network conditions similar to BBR and Cubic, using Pantheon traces, which we introduced above.

Model	Random	Original	Current
Aurora (small)	631.38	3195.15	3279.03
Aurora (mid)	631.38	3273.63	1380.22

Table 4: Aurora Model Rewards

C DBSCAN CLUSTERING SETTINGS

DBSCAN operates by identifying *core points* in given data. Core points have a prespecified minimum number of points in their neighborhood, specified by a given radius r. We set the minimum number of samples min_s to the number of points that make the cluster achieve the representation threshold τ_{rep}. For a low volume of specifications, we keep r as the minimum radius that can ideally contain the minimum number of points, if densely packed.

D ABLATIONS

To select the best thresholds and SpecTRA's parameters, we study the variation in the support and confidence of SpecTRA's specifications over the training observations with the various settings. Specifically, we consider the history length of the features used in the specifications, representation threshold τ_{rep}, the number of partitions (p) of the input space $\mathcal{X}$ along each dimension, and the maximum permissible number of outputs in each specification τ_{max}. Figures 2a and 2b show the quality of the specifications for ABR and CC respectively. We select those parameters for our main experiments that yield specifications with high support and confidence over all the references, as observed in this ablation study. We select history = 3 (previous 3 download times will be used in specifications), $\tau_{rep} = 0.01, p = 100, \tau_{max} = 5$ for ABR and history = 4 (previous 4 observed features will be used in specifications), $\tau_{rep} = 0.01, p = 50, \tau_{max} = 2$ for CC.

E SPECTRA'S GENERATED SPECIFICATION SET FOR ABR

The following specifications 3 were generated by SpecTRA for ABR, to be used in conjunction.

Specifications 3: Conjunctive specifications for ABR. (BS: Buffer Size, DT[$-i$]: i^{th} last download time, BR: Bit Rate)

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.15, 0.66]$,
$DT[-2] \in [0.15, 0.79]$,
$DT[-3] \in [0.54, 1.05]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 2850.0\}$

1

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.15, 1.05]$,
$DT[-2] \in [0.02, 0.79]$,
$DT[-3] \in [0.15, 1.05]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 2850.0, 4300.0\}$

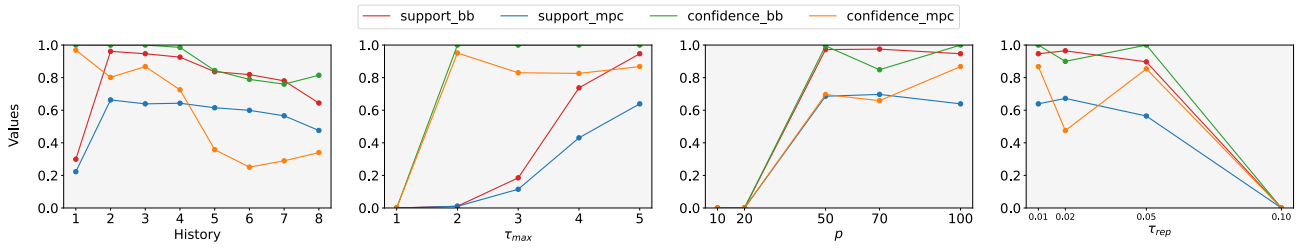
2

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.28, 0.66]$,
$DT[-2] \in [0.15, 0.66]$,
$DT[-3] \in [0.54, 0.92]$
Postcondition
$BR \in \{300.0, 750.0, 2850.0\}$

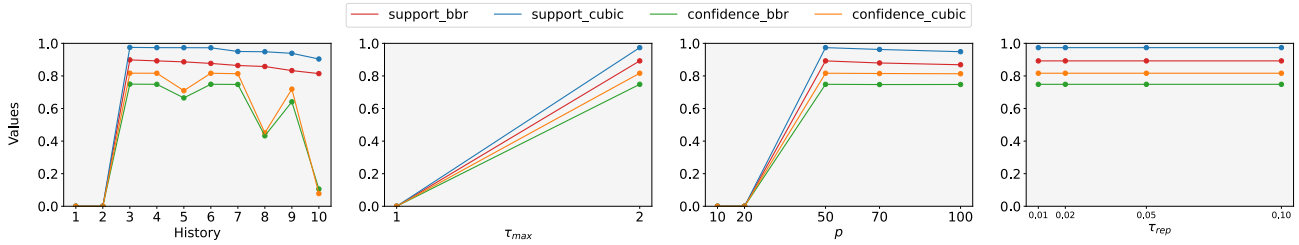
3

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.28, 0.66]$,
$DT[-2] \in [0.15, 0.79]$,
$DT[-3] \in [0.41, 1.05]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 4300.0\}$

4



(a) ABR



(b) CC

Figure 2: Ablation study for hyperparameters affecting the specifications

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.28, 0.66]$,
$DT[-2] \in [0.28, 0.66]$,
$DT[-3] \in [0.54, 0.92]$
Postcondition
$BR \in \{300.0, 750.0\}$

5

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.28, 0.66]$,
$DT[-2] \in [0.54, 1.05]$,
$DT[-3] \in [0.28, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 2850.0, 4300.0\}$

6

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.28, 0.66]$,
$DT[-2] \in [0.66, 1.05]$,
$DT[-3] \in [0.28, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0\}$

7

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.28, 1.05]$,
$DT[-2] \in [0.15, 0.79]$,
$DT[-3] \in [0.15, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 2850.0\}$

8

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.54, 1.05]$,
$DT[-2] \in [0.15, 0.66]$,
$DT[-3] \in [0.15, 0.54]$
Postcondition
$BR \in \{300.0, 750.0, 1850.0\}$

9

Precondition
$BS \in [0.4, 0.5]$,
$DT[-1] \in [0.54, 1.05]$,
$DT[-2] \in [0.15, 0.66]$,
$DT[-3] \in [0.15, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0\}$

10

Precondition
$BS \in [0.4, 0.54]$,
$DT[-1] \in [0.15, 0.66]$,
$DT[-2] \in [0.15, 0.79]$,
$DT[-3] \in [0.41, 1.05]$
Postcondition
$BR \in \{300.0, 750.0, 1850.0, 2850.0\}$

11

Precondition
$BS \in [0.4, 0.54]$,
$DT[-1] \in [0.54, 1.05]$,
$DT[-2] \in [0.15, 0.66]$,
$DT[-3] \in [0.15, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1850.0, 2850.0, 4300.0\}$

12

Precondition
$BS \in [0.4, 1.09]$,
$DT[-1] \in [0.02, 1.18]$,
$DT[-2] \in [0.02, 1.18]$,
$DT[-3] \in [0.02, 1.18]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 1850.0, 4300.0\}$

13

Precondition
$BS \in [0.4, 1.48]$,
$DT[-1] \in [0.02, 1.18]$,
$DT[-2] \in [0.02, 1.18]$,
$DT[-3] \in [0.02, 1.18]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 1850.0, 2850.0\}$

14

Precondition
$BS \in [0.5, 0.61]$,
$DT[-1] \in [0.54, 0.92]$,
$DT[-2] \in [0.15, 0.54]$,
$DT[-3] \in [0.15, 0.54]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 2850.0, 4300.0\}$

15

Precondition
$BS \in [0.61, 0.75]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.41, 1.05]$,
$DT[-3] \in [0.15, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1850.0\}$

16

Precondition
$BS \in [0.61, 0.78]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.41, 1.05]$,
$DT[-3] \in [0.15, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1850.0, 4300.0\}$

17

Precondition
$BS \in [0.61, 0.78]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.41, 1.18]$,
$DT[-3] \in [0.15, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 4300.0\}$

18

Precondition
$BS \in [0.61, 0.82]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.41, 1.05]$,
$DT[-3] \in [0.15, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1850.0, 2850.0, 4300.0\}$

19

Precondition
$BS \in [0.61, 1.06]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.02, 1.18]$,
$DT[-3] \in [0.15, 1.18]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 2850.0, 4300.0\}$

20

Precondition
$BS \in [0.64, 0.75]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.15, 0.54]$,
$DT[-3] \in [0.54, 1.18]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0\}$

21

Precondition
$BS \in [0.64, 0.78]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.15, 0.54]$,
$DT[-3] \in [0.54, 0.92]$
Postcondition
$BR \in \{300.0, 750.0, 1850.0, 2850.0\}$

22

Precondition
$BS \in [0.75, 0.82]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.15, 0.54]$,
$DT[-3] \in [0.54, 0.92]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0\}$

23

Precondition
$BS \in [0.78, 0.92]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.54, 0.92]$,
$DT[-3] \in [0.28, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0\}$

24

Precondition
$BS \in [0.82, 0.89]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.54, 0.92]$,
$DT[-3] \in [0.28, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1850.0, 2850.0\}$

25

Precondition
$BS \in [0.82, 0.96]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.41, 0.92]$,
$DT[-3] \in [0.15, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 2850.0\}$

26

Precondition
$BS \in [0.85, 0.96]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.02, 0.41]$,
$DT[-3] \in [0.28, 0.66]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0\}$

27

Precondition
$BS \in [0.85, 1.06]$,
$DT[-1] \in [0.02, 0.41]$,
$DT[-2] \in [0.02, 0.41]$,
$DT[-3] \in [0.54, 1.05]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0\}$

28

Precondition
$BS \in [1.09, 1.2]$,
$DT[-1] \in [0.02, 0.54]$,
$DT[-2] \in [0.02, 0.54]$,
$DT[-3] \in [0.41, 0.79]$
Postcondition
$BR \in \{300.0, 750.0, 1200.0, 1850.0\}$

29

Precondition
$BS \in [1.09, 1.23]$,
$DT[-1] \in [0.41, 0.79]$,
$DT[-2] \in [0.15, 0.66]$,
$DT[-3] \in [0.15, 0.54]$
Postcondition
$BR \in \{750.0, 1200.0, 1850.0, 2850.0, 4300.0\}$

30