Certified Robustness to Programmable Transformations in LSTMs

Yuhao Zhang, Aws Albarghouthi*, Loris D'Antoni
Department of Computer Science, University of Wisconsin-Madison, USA.
{yuhaoz, aws, loris}@cs.wisc.edu

Abstract

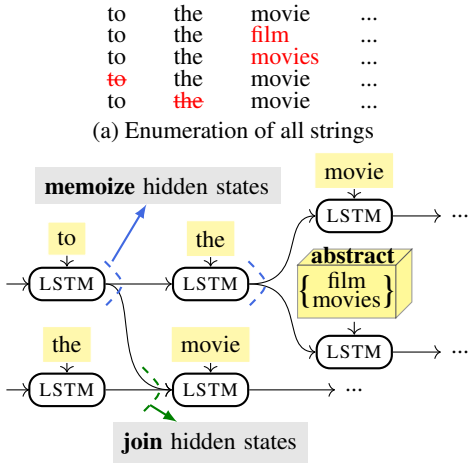
Deep neural networks for natural language processing are fragile in the face of adversarial examples—small input perturbations, like synonym substitution or word duplication, which cause a neural network to change its prediction. We present an approach to *certifying* the robustness of LSTMs (and extensions of LSTMs) and *training* models that can be efficiently certified. Our approach can certify robustness to intractably large perturbation spaces defined *programmatically* in a language of string transformations. Our evaluation shows that (1) our approach can train models that are more robust to combinations of string transformations than those produced using existing techniques; (2) our approach can show high certification accuracy of the resulting models.

1 Introduction

Adversarial examples are small perturbations of an input that fool a neural network into changing its prediction (Carlini and Wagner, 2017; Szegedy et al., 2014). In NLP, adversarial examples involve modifying an input string by, for example, replacing words with synonyms, deleting stop words, inserting words, etc. (Ebrahimi et al., 2018; Li et al., 2019; Zhang et al., 2019).

Ideally, a *defense* against adversarial examples in NLP tasks should fulfill the following desiderata: (1) Handle **recursive models**, like LSTMs and extensions thereof, which are prevalent in NLP. (2) Construct **certificates** (proofs) of robustness. (3) Defend against **arbitrary string transformations**, like combinations of word deletion, insertion, etc.

It is quite challenging to fulfill all three desiderata; indeed, existing techniques are forced to make tradeoffs. For instance, the theoretical insights underlying a number of certification approaches are intimately tied to symbol substitution (Jia et al., 2019; Huang et al., 2019; Ye et al., 2020; Xu et al., 2020; Dong et al., 2021), and some techniques cannot handle recursive models (Huang et al., 2019; Zhang et al., 2020). On the other hand, techniques that strive to be robust to arbitrary string transformations achieve this at the expense of certification (Zhang et al., 2020; Ebrahimi et al., 2018).



(a) Enumeration of all strings

(b) ARC: Abstract Recursive Certification

Figure 1: An illustration of our approach.

In this paper, we ask: Can we develop a *certified* defense to *arbitrary string transformations* that applies to *recursive neural networks*?

Our approach. Certifying robustness involves proving that a network's prediction is the same no matter how a given input string is perturbed. We assume that the *perturbation space* is defined as a program describing a set of possible string transformations (Zhang et al., 2020)—e.g., *if you see the word "movie", replace it with "film" or "movies"*. Such transformations can succinctly define a perturbation space that is exponentially large in the length of the input; so, certification by enumerating the perturbation space is generally impractical.

*Author's name in native alphabet: أوس البرغوثي

We present ARC (Abstract Recursive Certification), an approach for *certifying* robustness to programmatically defined perturbation spaces. ARC can be used within an adversarial training loop to *train robust models*. We illustrate the key ideas behind ARC through a simple example. Consider the (partial) input sentence *to the movie...*, and say we are using an LSTM for prediction. Say we have two string transformations: (**T1**) If you see the word *movie*, you can replace it with *film* or *movies*. (**T2**) If you see the word *the* or *to*, you can delete it. ARC avoids enumerating the large perturbation space (Fig. 1(a)) using two key insights.

Memoization: ARC exploits the recursive structure of LSTM networks, and their extensions (BiLSTMs, TreeLSTMs), to avoid recomputing intermediate hidden states. ARC *memoizes* hidden states of prefixes shared across multiple sequences in the perturbation space. For example, the two sentences *to the movie...* and *to the film....* share the prefix *to the*, and therefore we memoize the hidden state after the word *the*, as illustrated in Fig. 1(b) with dashed blue lines. The critical challenge is characterizing which strings share prefixes without having to explicitly explore the perturbation space.

Abstraction: ARC uses *abstract interpretation* (Cousot and Cousot, 1977) to symbolically represent sets of perturbed strings, avoiding a combinatorial explosion. Specifically, ARC represents a set of strings as a hyperrectangle in a $\mathbb{R}^n$ and propagates the hyperrectangle through the network using interval arithmetic (Gehr et al., 2018; Gowal et al., 2019).This idea is illustrated in Fig. 1(b), where the words *film* and *movies* are represented as a hyperrectangle. By *joining* hidden states of different sentences (a common idea in program analysis), ARC can perform certification efficiently.

Memoization and abstraction enable ARC to efficiently certify robustness to very large perturbation spaces. Note that ARC subsumes Xu et al. (2020) because ARC can certify arbitrary string transformations, while Xu et al. (2020) only works on word substitutions.

Contributions. We make the following contributions: (1) We present ARC, an approach for training certifiably robust recursive neural networks. We demonstrate our approach on LSTMs, BiLSTMs, and TreeLSTMs. (2) We present a novel application of abstract interpretation to symbolically capture a large space of strings, defined programmatically, and propagate it through a recursive network (Section 4). (3) Our evaluation shows that ARC can train models that are more robust to arbitrary perturbation spaces than those produced by existing techniques; ARC can show high certification accuracy of the resulting models; and ARC can certify robustness to attacks (transformations) that are out-of-scope for existing techniques (Section 5).

Table 1: ARC compared to other approaches. Ye et al. (2020) provide probabilistic certificates.

Approach	LSTM	Cert	Transf.
Ebrahimi et al. (2018)	✓	✗	*arbitrary*
Ko et al. (2019a)	✓	✓	l_p norm
Huang et al. (2019)	✗	✓	substitution
Jia et al. (2019)	✓	✓	substitution
Zhang et al. (2020)	✗	✗	*arbitrary*
Ye et al. (2020)	✓	$\mathbb{P}$	substitution
Xu et al. (2020)	✓	✓	substitution
Dong et al. (2021)	✓	✗	substitution
ARC (*this paper*)	✓	✓	*arbitrary*

2 Related Work

Table 1 compares ARC to most related approaches.

Certification of robustness in NLP. See Li et al. (2020) for a survey of robust training. Some works focus on certifying the l_p norm ball of each word embedding for LSTMs (Ko et al., 2019b; Jacoby et al., 2020) and transformers (Shi et al., 2020). Others focus on certifying word substitutions for CNNs (Huang et al., 2019) and LSTMs (Jia et al., 2019; Xu et al., 2020), and word deletions for the decomposable attention model (Welbl et al., 2020). Existing techniques rely on abstract interpretation, such as IBP (Gowal et al., 2019) and CROWN (Zhang et al., 2018). We focus on certifying the robustness of LSTM models (including TreeLSTMs) to a programmable perturbation space, which is out-of-scope for existing techniques. Note that Xu et al. (2020) also uses memoization and abstraction to certify, but ARC subsumes Xu et al. (2020) because ARC can certify arbitrary string transformations. We use IBP, but our approach can use other abstract domains, such as zonotopes (Gehr et al., 2018).

SAFER (Ye et al., 2020) is a model-agnostic approach that uses *randomized smoothing* (Cohen et al., 2019) to give probabilistic certificates of robustness to word substitution. Our approach gives a non-probabilistic certificate and can handle arbitrary perturbation spaces beyond substitution.

Robustness techniques in NLP. Adversarial train-

ing is an empirical defense method that can improve the robustness of models by solving a *robust-optimization* problem (Madry et al., 2018), which minimizes worst-case (adversarial) loss. Some techniques in NLP use adversarial attacks to compute a lower bound on the worst-case loss (Ebrahimi et al., 2018; Michel et al., 2019). ASCC (Dong et al., 2021) overapproximates the word substitution attack space by a convex hull where a lower bound on the worst-case loss is computed using gradients. Other techniques compute upper bounds on adversarial loss using abstract interpretation (Gowal et al., 2019; Mirman et al., 2018). Huang et al. (2019) and Jia et al. (2019); Xu et al. (2020) used abstract interpretation to train CNN and LSTM models against word substitutions. A3T (Zhang et al., 2020) trains robust CNN models against a programmable perturbation space by combining adversarial training and abstraction. Our approach uses abstract interpretation to train robust LSTMs against programmable perturbation spaces as defined in Zhang et al. (2020).

3 Robustness Problem and Preliminaries

We consider a classification setting with a neural network F_θ with parameters θ, trained on samples from domain $\mathcal{X}$ and labels from $\mathcal{Y}$. The domain $\mathcal{X}$ is a set of strings over a finite set of *symbols* Σ (e.g., English words or characters), i.e., $\mathcal{X} = \Sigma^*$. We use $\mathbf{x} \in \Sigma^*$ to denote a string; $x_i \in \Sigma$ to denote the ith element of the string; $\mathbf{x}_{i:j}$ to denote the substring $x_i, \ldots, x_j$; ϵ to denote the empty string; and $\text{LEN}_\mathbf{x}$ to denote the length of the string.

Robustness to string transformations. A *perturbation space* S is a function in $\Sigma^* \to 2^{\Sigma^*}$, i.e., S takes a string $\mathbf{x}$ and returns a set of possible perturbed strings obtained by modifying $\mathbf{x}$. Intuitively, $S(\mathbf{x})$ denotes a set of strings that are semantically *similar* to $\mathbf{x}$ and therefore should receive the same prediction. We assume $\mathbf{x} \in S(\mathbf{x})$.

Given string $\mathbf{x}$ with label y and a perturbation space S, We say that a neural network F_θ is S-*robust* on $(\mathbf{x}, y)$ iff

$$\forall \mathbf{z} \in S(\mathbf{x}).\, F_\theta(\mathbf{z}) = y \tag{1}$$

Our primary goal in this paper is to *certify*, or prove, S-robustness (Eq 1) of the neural network for a pair $(\mathbf{x}, y)$. Given a certification approach, we can then use it within an adversarial training loop to yield certifiably robust networks.

Robustness certification. We will certify S-robustness by solving an *adversarial loss* objective:

$$\max_{\mathbf{z} \in S(\mathbf{x})} L_\theta(\mathbf{z}, y) \tag{2}$$

where we assume that the loss function L_θ is < 0 when $F_\theta(\mathbf{z}) = y$ and $\geqslant 0$ when $F_\theta(\mathbf{z}) \neq y$. Therefore, if we can show that the solution to the above problem is < 0, then we have a certificate of S-robustness.

Certified training. If we have a procedure to compute adversarial loss, we can use it for *adversarial training* by solving the following *robust optimization* objective (Madry et al., 2018), where $\mathcal{D}$ is the data distribution:

$$\arg\min_\theta \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\max_{\mathbf{z} \in S(\mathbf{x})} L_\theta(\mathbf{z}, y) \right] \tag{3}$$

3.1 Programmable Perturbation Spaces

In our problem definition, we assumed an arbitrary perturbation space S. We adopt the recently proposed specification language (Zhang et al., 2020) to define S *programmatically* as a set of string transformations. The language is very flexible, allowing the definition of a rich class of transformations as *match* and *replace* functions.

Single transformations. A string transformation T is a pair (φ, f), where $\varphi : \Sigma^s \to \{0, 1\}$ is the *match* function, a Boolean function that specifies the substrings (of length s) to which the transformation can be applied; and $f : \Sigma^s \to 2^{\Sigma^t}$ is the *replace* function, which specifies how the substrings matched by φ can be replaced (with strings of length t). We will call s and t the size of the *domain* and *range* of transformation T, respectively.

Example 3.1. *In all examples, the set of symbols Σ is English words. So, strings are English sentences. Let T_{del} be a string transformation that deletes the stop words "*to*" and "*the*". Formally, $T_{\text{del}} = (\varphi_{\text{del}}, f_{\text{del}})$, where $\varphi_{\text{del}} : \Sigma^1 \mapsto \{0, 1\}$ and $f_{\text{del}} : \Sigma^1 \to 2^{\Sigma^0}$ are:*

$$\varphi_{\text{del}}(x) = \begin{cases} 1, & x \in \{\text{“to”, “the”}\} \\ 0, & \textit{otherwise} \end{cases}, \quad f_{\text{del}}(x) = \{\epsilon\},$$

*Let T_{sub} be a transformation substituting the word "*movie*" with "*movies*" or "*film*". Formally, $T_{\text{sub}} = (\varphi_{\text{sub}}, f_{\text{sub}})$, where $\varphi_{\text{sub}} : \Sigma^1 \mapsto \{0, 1\}$ and $f_{\text{sub}} : \Sigma^1 \to 2^{\Sigma^1}$ are:*

$$\varphi_{\text{sub}}(x) = \begin{cases} 1, & x = \text{“movie”} \\ 0, & \textit{otherwise} \end{cases}, f_{\text{sub}}(x) = \left\{ \begin{matrix} \text{“film”}, \\ \text{“movies”} \end{matrix} \right\}$$

Defining perturbation spaces. We can compose different string transformation to construct perturbation space S:

$$S = \{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}, \tag{4}$$

where each T_i denotes a string transformation that can be applied *up to* $\delta_i \in \mathbb{N}$ times. Note that the transformations can be applied whenever they match a *non-overlapping* set of substrings and are then transformed in parallel. We illustrate with an example and refer to Zhang et al. (2020) for formal semantics.

Example 3.2. *Let* $S = \{(T_{\text{del}}, 1), (T_{\text{sub}}, 1)\}$ *be a perturbation space that applies* T_{del} *and* T_{sub} *to the given input sequence up to once each. If* $\mathbf{x} =$"to the movie"*, a subset of the perturbation space* $S(\mathbf{x})$ *is shown in Fig. 1(a).*

Decomposition. $S = \{(T_i, \delta_i)\}_i$ can be decomposed into $\prod(\delta_i + 1)$ subset perturbation spaces by considering all smaller combinations of δ_i. We denote the decomposition of perturbation space S as DEC_S, and exemplify below:

Example 3.3. $S_1 = \{(T_{\text{del}}, 2)\}$ *can be decomposed to a set of three perturbation spaces* $\text{DEC}_{S_1} = \{\varnothing, \{(T_{\text{del}}, 1)\}, \{(T_{\text{del}}, 2)\}\}$*, while* $S_2 = \{(T_{\text{del}}, 1), (T_{\text{sub}}, 1)\}$ *can be decomposed to a set of four perturbation spaces*

$$\text{DEC}_{S_2} = \{\varnothing, \{(T_{\text{del}}, 1)\}, \{(T_{\text{sub}}, 1)\}, \{(T_{\text{del}}, 1), (T_{\text{sub}}, 1)\}\}$$

where $\varnothing$ *is the perturbation space with no transformations, i.e., if* $S = \varnothing$*, then* $S(\mathbf{x}) = \{\mathbf{x}\}$ *for any string* $\mathbf{x}$*.*

We use notation $S_{k\downarrow}$ to denote S after reducing δ_k by 1; therefore, $S_{k\downarrow} \in \text{DEC}_S$.

3.2 The LSTM Cell

We focus our exposition on LSTMs. An LSTM cell is a function, denoted LSTM, that takes as input a symbol x_i and the previous *hidden state* and *cell state*, and outputs the hidden state and the cell state at the current time step. For simplicity, we use h_i to denote the concatenation of the hidden and cell states at the ith time step, and simply refer to it as the *state*. Given string $\mathbf{x}$, we define h_i as follows:

$$h_i = \text{LSTM}(x_i, h_{i-1}) \quad h_i, h_{i-1} \in \mathbb{R}^d,$$

where $h_0 = 0^d$ and d is the dimensionality of the state. For a string $\mathbf{x}$ of length n, we say that h_n is the *final state*.

For a string $\mathbf{x}$ and state h, We use $\text{LSTM}(\mathbf{x}, h)$ to denote

$$\text{LSTM}(x_{\text{LEN}_\mathbf{x}}, \text{LSTM}(\ldots, \text{LSTM}(x_1, h)\ldots))$$

E.g., $\text{LSTM}(\mathbf{x}, h_0)$ is the final state of the LSTM applied to $\mathbf{x}$.

4 ARC: Abstract Recursive Certification

In this section, we present our technique for proving S-robustness of an LSTM on $(\mathbf{x}, y)$. Formally, we do this by computing the adversarial loss, $\max_{\mathbf{z} \in S(\mathbf{x})} L_\theta(\mathbf{z}, y)$. Recall that if the solution is < 0, then we have proven S-robustness. To solve adversarial loss optimally, we effectively need to evaluate the LSTM on all of $S(\mathbf{x})$ and collect all final states:

$$F = \{\text{LSTM}(\mathbf{z}, h_0) \mid \mathbf{z} \in S(\mathbf{x})\} \tag{5}$$

Computing F precisely is challenging, as $S(\mathbf{x})$ may be prohibitively large. Therefore, we propose to compute a superset of F, which we will call $\widehat{F}$. This superset will therefore yield an upper bound on the adversarial loss. We prove S-robustness if the upper bound is < 0.

To compute $\widehat{F}$, we present two key ideas that go hand-in-hand: In Section 4.1, we observe that strings in the perturbation space share common prefixes, and therefore we can *memoize* hidden states to reduce the number of evaluations of LSTM cells—a form of dynamic programming. We carefully derive the set of final states F as a system of memoizing equations. The challenge of this derivation is characterizing which strings share common prefixes without explicitly exploring the perturbation space. In Section 4.2, we apply abstract interpretation to efficiently and soundly solve the system of memoizing equations, thus computing an overapproximation $\widehat{F} \supseteq F$.

4.1 Memoizing Equations of Final States

Tight Perturbation Space. Given a perturbation space S, we shall use $S^=$ to denote the *tight* perturbation space where each transformation T_j in S is be applied **exactly** δ_j times.

Think of the set of all strings in a perturbation space as a tree, like in Fig. 1(b), where strings that share prefixes share LSTM states. We want to characterize a subset $H^S_{i,j}$ of LSTM states at the ith layer where the perturbed prefixes have had *all* transformations in a space S applied on the original prefix $\mathbf{x}_{1:j}$.

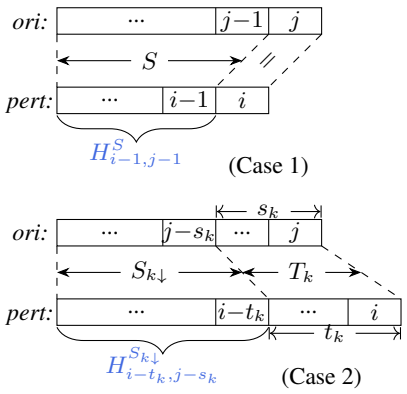


Figure 2: Illustration of two cases of Eq 8.

We formally define $H^S_{i,j}$ as follows:

$$H^S_{i,j} = \{\textsc{lstm}(\mathbf{z}, h_0) \mid \mathbf{z} \in S^{=}(\mathbf{x}_{1:j}) \wedge \textsc{len}_{\mathbf{z}} = i\} \quad (6)$$

By definition, the base case $H^{\varnothing}_{0,0} = \{0^d\}$.

Example 4.1. *Let* $\mathbf{x}$ = "to the movie". *Then,* $H^{\{(T_{\text{del}},1)\}}_{1,2} = \{\textsc{lstm}(\text{"the"}, h_0), \textsc{lstm}(\text{"to"}, h_0)\}$. *These states result from deleting the first and second words of the prefix* "to the", *respectively. We also have* $H^{\varnothing}_{2,2} = \{\textsc{lstm}(\text{"to the"}, h_0)\}$.

The set of final states of strings in $S^{=}(\mathbf{x})$ is

$$\bigcup_{i \geq 0} H^S_{i,\textsc{len}_{\mathbf{x}}}. \quad (7)$$

Memoizing equation. We now demonstrate how to rewrite Eq 6 by explicitly applying the transformations defining the perturbation space S. Notice that each $H^S_{i,j}$ comes from two sets of strings: (1) strings whose suffix (the last character) is not perturbed by any transformations (the first line of Eq 8), and (2) strings whose suffix is perturbed by $T_k = (\varphi_k, f_k)$ (the second line of Eq 8), as illustrated in Fig 2. Thus, we derive the final equation and then immediately show an example:

$$H^S_{i,j} = \{\textsc{lstm}(x_j, h) \mid h \in H^S_{i-1,j-1}\} \cup \bigcup_{\substack{1 \leqslant k \leqslant |S| \\ \varphi_k(\mathbf{x}_{a:b})=1}} \{\textsc{lstm}(\mathbf{z}, h) \mid \mathbf{z} \in f_k(\mathbf{x}_{a:b}), h \in H^{S_{k\downarrow}}_{i-t_k,j-s_k}\} \quad (8)$$

where $a{=}j{-}s_k+1$ and $b{=}j$.

We compute Eq 8 in a bottom-up fashion, starting from $H^{\varnothing}_{0,0} = \{0^d\}$ and increasing i, j and considering every possible perturbation space in the decomposition of S, DEC_S.

Lemma 4.1. *Eq 8 and Eq 6 are equivalent.*

Example 4.2. *Consider computing* $H^{\{(T_{\text{del}},1)\}}_{1,2}$. *We demonstrate how to derive states from Eq 8:*

$$H^{\{(T_{\text{del}},1)\}}_{1,2} = \{\textsc{lstm}(\text{"the"}, h) \mid h \in H^{\{(T_{\text{del}},1)\}}_{0,1}\} \cup \{\textsc{lstm}(\mathbf{z}, h) \mid \mathbf{z} \in f_{\text{del}}(\text{"the"}), h \in H^{\varnothing}_{1,1}\} \quad (9)$$

Assume $H^{\varnothing}_{1,1} = \{\textsc{lstm}(\text{"to"}, h_0)\}$ *and* $H^{\{(T_{\text{del}},1)\}}_{0,1} = \{h_0\}$ *are computed in advance. The first line of Eq 9 evaluates to* $\{\textsc{lstm}(\text{"the"}, h_0)\}$, *which corresponds to deleting the first word of the prefix "to the". Because* $\mathbf{z}$ *can only be an empty string, the second line of Eq 9 evaluates to* $\{\textsc{lstm}(\text{"to"}, h_0)\}$, *which corresponds to deleting the second word of "to the". The dashed green line in Fig. 1(b) shows the computation of Eq 9.*

Defining Final States using Prefixes. Finally, we compute the set of final states, F, by considering all perturbation spaces in the decomposition of S.

$$F = \bigcup_{S' \in \text{DEC}_S} \bigcup_{i \geq 0} H^{S'}_{i,\textsc{len}_{\mathbf{x}}} \quad (10)$$

Theorem 4.1. *Eq 10 is equivalent to Eq 5.*

Example 4.3. *Let* $S = \{(T_{\text{del}}, 1), (T_{\text{sub}}, 1)\}$ *and* $\mathbf{x}$ = *"to the movie". F is the union of four final states,* $H^{\varnothing}_{3,3}$ *(no transformations),* $H^{\{(T_{\text{del}},1)\}}_{2,3}$ *(exactly 1 deletion),* $H^{\{(T_{\text{sub}},1)\}}_{3,3}$ *(exactly 1 substitution), and* $H^{\{(T_{\text{del}},1),(T_{\text{sub}},1)\}}_{2,3}$ *(exactly 1 deletion and 1 substitution).*

4.2 Abstract Memoizing Equations

Memoization avoids recomputing hidden states, but it still incurs a combinatorial explosion. We employ abstract interpretation (Cousot and Cousot, 1977) to solve the equations efficiently by overapproximating the set F. See Albarghouthi (2021) for details on abstractly interpreting neural networks.

Abstract Interpretation. The *interval domain*, or *interval bound propagation*, allows us to evaluate a function on an infinite set of inputs represented as a hyperrectangle in $\mathbb{R}^n$.

Interval domain. We define the interval domain over scalars—the extension to vectors is standard. We will use an interval $[l, u] \subset \mathbb{R}$, where $l, u \in \mathbb{R}$ and $l \leqslant u$, to denote the set of all real numbers between l and u, inclusive.

For a finite set $X \subset \mathbb{R}$, the *abstraction* operator gives the tightest interval containing X, as follows: $\alpha(X) = [\min(X), \max(X)]$. Abstraction allows us to compactly represent a large set of strings.

Example 4.4. *Suppose the words* x_1 = "movie" *and* x_2 = "film" *have the 1D embedding* 0.1 *and* 0.15*, respectively. Then,* $\alpha(\{x_1, x_2\}) = [0.1, 0.15]$. *For* n*-dimensional embeddings, we simply compute an abstraction of every dimension, producing a vector of intervals.*

The *join* operation, $\sqcup$, produces the smallest interval containing two intervals: $[l, u] \sqcup [l', u'] = [\min(l, l'), \max(u, u')]$. We will use joins to merge hidden states resulting from different strings in the perturbation space (recall Fig. 1(b)).

Example 4.5. *Say we have two sets of 1D LSTM states represented as intervals,* $[1, 2]$ *and* $[10, 12]$. *Then* $[1, 2] \sqcup [10, 12] = [1, 12]$. *Note that* $\sqcup$ *is an overapproximation of* $\cup$*, introducing elements in neither interval.*

Interval transformers. To evaluate a neural network on intervals, we lift neural-network operations to interval arithmetic—*abstract transformers*. For a function g, we use $\widehat{g}$ to denote its abstract transformer. We use the transformers proposed by Gehr et al. (2018); Jia et al. (2019). We illustrate transformers for addition and any monotonically increasing function $g : \mathbb{R} \to \mathbb{R}$ (e.g., ReLU, tanh).

$$[l, u] \widehat{+} [l', u'] = [l + l', u + u'], \qquad \widehat{g}([l, u]) = [g(l), g(u)]$$

Note how, for monotonic functions g, the abstract transformer $\widehat{g}$ simply applies g to the lower and upper bounds.

Example 4.6. *When applying to the ReLU function,* $\widehat{\text{relu}}([-1, 2]) = [\text{relu}(-1), \text{relu}(2)] = [0, 2]$.

An abstract transformer $\widehat{g}$ must be *sound*: for any interval $[l, u]$ and $x \in [l, u]$, we have $g(x) \in \widehat{g}([l, u])$. We use $\widehat{\text{LSTM}}$ to denote an abstract transformer of an LSTM cell. It takes an interval of symbol embeddings and an interval of states. We use the definition of $\widehat{\text{LSTM}}$ given by Jia et al. (2019).

Abstract Memoizing Equations. We now show how to solve Eq 8 and Eq 10 using abstract interpretation. We do this by rewriting the equations using operations over intervals. Let $\widehat{H}^{\varnothing}_{0,0} = \alpha(\{0^d\})$, then

$$\widehat{H}^{S}_{i,j} = \widehat{\text{LSTM}}(\alpha(\{x_j\}), \widehat{H}^{S}_{i-1,j-1}) \sqcup \bigsqcup_{\substack{1 \leqslant k \leqslant |S| \\ \varphi_k(\mathbf{x}_{a:b})=1}} \widehat{\text{LSTM}}(\alpha(f_k(\mathbf{x}_{a:b})), \widehat{H}^{S_{k\downarrow}}_{i-t_k, j-s_k})$$

$$\widehat{F} = \bigsqcup_{S' \in \text{DEC}_S} \bigsqcup_{i \geq 0} \widehat{H}^{S'}_{i, \text{LEN}_{\mathbf{x}}}$$

where a and b are the same in Eq 8.

The two key ideas are (1) representing sets of possible LSTM inputs abstractly as intervals, using α; and (2) joining intervals of states, using $\sqcup$. These two ideas ensure that we efficiently solve the system of equations, producing an overapproximation $\widehat{F}$.

The above abstract equations give us a compact overapproximation of F that can be computed with a number of steps that is linear in the length of the input. Even though we can have $O(\text{LEN}^2_{\mathbf{x}})$ number of $H^S_{i,j}$ for a given S, only $O(\text{LEN}_{\mathbf{x}})$ number of $H^S_{i,j}$ are non-empty. This property is used in Theorem 4.2 and will be proved in the appendix.

Theorem 4.2. *(Soundness & Complexity)* $F \subseteq \widehat{F}$ *and the number of LSTM cell evaluations needed to compute* $\widehat{F}$ *is* $O(\text{LEN}_{\mathbf{x}} \cdot n \cdot \prod_{i=1}^{n} \delta_i)$.

For practical perturbations spaces (see Section 5), the quantity $n \prod_{i=1}^{n} \delta_i$ is typically small and can be considered constant.

Extension to Bi-LSTMs and Tree-LSTMs. A *Bi-LSTM* performs a forward and a backward pass on the input. The forward pass is the same as the forward pass in the original LSTM. For the backward pass, we reverse the input string $\mathbf{x}$, the input of the match function φ_i and the input/output of the replace function f_i of each transformation.

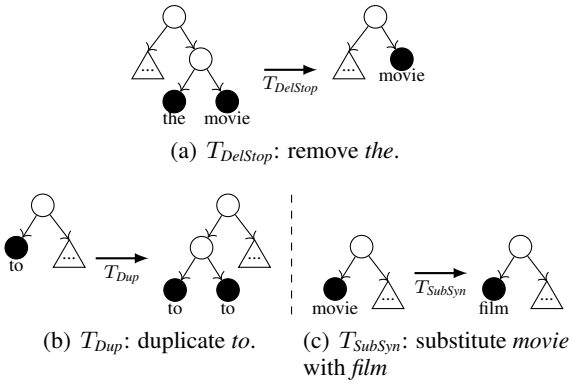
A *Tree-LSTM* takes trees as input. We can define the programmable perturbation space over trees in the same form of Eq 4, where T_i is a tree transformation. We show some examples of tree transformations in Fig 3. $T_{DelStop}$ (Fig 3(a)) removes a leaf node with a stop word in the tree. After removing, the sibling of the removed node becomes the new parent node. T_{Dup} (Fig 3(b)) duplicates a word in a leaf node by first removing the word and expanding the leaf node with two children, each of which contains the previous word. T_{SubSyn} (Fig 3(c)) substitutes a word in the leaf node with one of its synonyms.

We provide the formalization of ARC on Bi-LSTMs and Tree-LSTMs in the appendix.

5 Evaluation

We implemented ARC in PyTorch. The source code is available online[1] and provided in the supplementary materials.

[1] https://github.com/ForeverZyh/certified_lstms



(a) $T_{DelStop}$: remove *the*.

(b) T_{Dup}: duplicate *to*.

(c) T_{SubSyn}: substitute *movie* with *film*

Figure 3: Examples of tree transformations.

Table 2: String transformations.

Trans	Description
$T_{DelStop}$	**delete** a **stop word**, e.g., *and, the, a, to, of,...*
T_{Dup}	**duplicate** a word
T_{SubSyn}	**substitute** a word with one of its synonyms

Datasets. We use three datasets: *IMDB* (Maas et al., 2011), *Stanford Sentiment Treebank* (SST) (Socher et al., 2013), and SST2, a two-way class split of SST. IMDB and SST2 have reviews in sentence form and binary labels. SST has reviews in the *constituency parse tree* form and five labels.

Perturbation Spaces. Following Zhang et al. (2020), we create perturbation spaces by combining the transformations in Table 2, e.g., $\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$ removes up to two stop words and replaces up to two words with synonyms. We also design a domain-specific perturbation space S_{review} for movie reviews; e.g., one transformation in S_{review} can duplicate question or exclamation marks because they usually appear repeatedly in movie reviews. We provide the detailed definition and evaluation of S_{review} in the appendix.

For Tree-LSTMs, we consider the tree transformations exemplified in Fig 3 and described in Section 4.2.

Evaluation metrics. (1) *Accuracy* (Acc.) is the vanilla test accuracy. (2) *HotFlip accuracy* (HF Acc.) is the adversarial accuracy with respect to the HotFlip attack (Ebrahimi et al., 2018): for each point in the test set, we use HotFlip to generate the top-5 perturbed examples and test if the classifications are correct on all the 5 examples and the original example. (3) *Certified accuracy* (CF Acc.) is the percentage of points in the test set certified as S-robust (Eq 1) using ARC. (4) *Exhaustive accuracy* (EX Acc.) is the percentage of points in the test set that is S-robust (Eq 1). HotFlip accuracy is an upper bound of exhaustive accuracy; certified accuracy is a lower bound of exhaustive accuracy.

Table 3: Qualitative examples. The vanilla model incorrectly classifies the perturbed samples.

Original sample in SST2 dataset i was perplexed to watch it unfold with an astonishing lack of passion or uniqueness .	**-ve**
Perturbed sample in $\{(T_{\text{DelStop}}, 2), (T_{\text{SubSyn}}, 2)\}$ i was perplexed ~~to~~ watch it unfold with an astounding absence of passion or uniqueness .	**+ve**
Original sample in SST2 dataset this is pretty dicey material .	**-ve**
Perturbed sample in $\{(T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$ this becomes pretty pretty dicey material . .	**+ve**

Baselines. For training certifiable models against arbitrary string transformations, we compare ARC to (1) *Normal training* that minimizes the cross entropy. (2) *Data augmentation* that augments the dataset with random samples from the perturbation space. (3) *HotFlip (Ebrahimi et al., 2018)* adversarial training. And (4) *A3T (Zhang et al., 2020)* that trains robust CNNs.

For training certifiable models against word substitution, we compare ARC to (1) *Jia et al. (2019)* that trains *certifiably* robust (Bi)LSTMs. We call this CertSub in the rest of this section. (2) *ASCC (Dong et al., 2021)* that trains *empirically* robust (Bi)LSTMs. And (3) *Huang et al. (2019)* that trains *certifiably* robust CNNs (comparison results are in the appendix).

For certification, we compare ARC to (1) *POPQORN (Ko et al., 2019a)*, the state-of-the-art approach for certifying LSTMs. (2) *SAFER (Ye et al., 2020)* that provides probabilistic certificates to word substitution.

Xu et al. (2020) is a special case of ARC where the perturbation space only contains substitution. We provide theoretical comparison in the appendix.

5.1 Arbitrary Perturbation Spaces

Comparison to Data Augmentation & HotFlip. We use the three perturbation spaces in Table 4 and the domain-specific perturbation space S_{review} in Table 5.

ARC outperforms data augmentation and HotFlip in terms of EX Acc. and CF Acc. Table 4 shows the results of LSTM, Tree-LSTM, and Bi-LSTM models on the tree perturbation

Table 4: Results of LSTM (on SST2), Tree-LSTM (on SST), and Bi-LSTM (on SST2) for three perturbation spaces.

		$\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$				$\{(T_{Dup}, 2), (T_{SubSyn}, 2)\}$				$\{(T_{DelStop}, 2), (T_{Dup}, 2)\}$			
	Train	Acc.	HF Acc.	CF Acc.	EX Acc.	Acc.	HF Acc.	CF Acc.	EX Acc.	Acc.	HF Acc.	CF Acc.	EX Acc.
LSTM	Normal	**84.6**	71.9	4.6	68.9	84.6	64.0	0.5	55.1	**84.6**	73.7	1.2	65.2
	Data Aug.	84.0	77.0	5.5	74.4	**84.7**	70.2	0.3	61.5	84.5	75.4	0.4	68.3
	HotFlip	84.0	**78.7**	4.3	74.6	82.5	**75.9**	0.0	62.0	84.4	**80.6**	0.0	68.7
	ARC	82.5	77.8	**72.5**	**77.0**	80.2	70.0	**55.4**	**64.0**	82.6	78.6	**69.4**	**74.6**
TREE-LSTM	Normal	**50.3**	39.9	4.1	33.8	**50.3**	33.4	0.0	17.9	**50.3**	40.1	0.0	25.7
	Data Aug.	47.5	40.8	1.4	36.4	48.1	37.1	0.0	23.0	47.6	40.6	0.0	29.0
	HotFlip	49.5	43.4	1.6	38.4	48.7	**39.5**	0.0	29.0	49.5	42.7	0.0	32.1
	ARC	46.4	**43.4**	**30.9**	**41.9**	46.1	39.0	**17.1**	**37.6**	46.5	**43.8**	**19.2**	**40.0**
BI-LSTM	Normal	83.0	71.1	8.2	68.0	83.0	63.4	2.1	56.1	83.0	72.5	6.4	65.5
	Data Aug.	83.2	75.1	8.7	72.9	**83.5**	66.8	1.3	59.1	**84.6**	75.0	4.6	68.6
	HotFlip	**83.6**	**79.2**	9.2	73.4	82.8	**76.6**	0.1	55.5	83.5	**79.1**	0.0	55.7
	ARC	83.5	78.7	**70.9**	**77.5**	80.2	71.4	**59.8**	**66.4**	82.6	76.2	**66.2**	**71.8**

Table 5: Results of LSTM on SST2 dataset for S_{review}.

	S_{review}			
Train	Acc.	HF Acc.	CF Acc.	EX Acc.
Normal	**83.9**	72.4	21.0	71.0
Data Aug.	79.2	72.5	30.5	71.7
HotFlip	79.6	74.3	34.5	72.9
ARC	82.3	**78.1**	**74.2**	**77.1**

spaces. Table 5 shows the results of LSTM models on the domain-specific perturbation space S_{review}. ARC has significantly higher EX Acc. than normal training ($+8.1, +14.0, +8.7$ on average), data augmentation ($+4.2, +10.4, +5.0$), and HotFlip ($+3.6, +6.7, +10.4$) for LSTM, Tree-LSTM, and Bi-LSTM respectively.

Models trained with ARC have a relatively high CF Acc. (53.6 on average). Data augmentation and HotFlip result in models not amenable to certification—in some cases, almost nothing in the test set can be certified.

ARC produces more robust models at the expense of accuracy. Other robust training approaches like CertSub and A3T also exhibit this trade-off. However, as we will show next, ARC retains higher accuracy than these approaches.

Comparison to A3T. A3T degenerates to HotFlip training on $\{(T_{DelStop}, 2), (T_{Dup}, 2)\}$, so we do not use this perturbation space.

The LSTMs trained using ARC are more robust than the CNNs trained by A3T for both perturbation spaces; ARC can certify the robustness of models while A3T cannot. Table 6 shows that ARC results in models with higher accuracy ($+2.3$ and $+0.3$), HF Acc. ($+5.9$ and $+1.7$), and EX Acc. ($+6.8$ and $+6.3$) than those produced by A3T. ARC can certify the trained models while A3T cannot.

5.2 Experiments on Word Substitution

We compare to works *limited* to word substitution.

Comparison to CertSub and ASCC. We choose two perturbation spaces, $\{(T_{SubSyn}, 1)\}$ and $\{(T_{SubSyn}, 2)\}$. We train one model per perturbation space using ARC under the same experimental setup of CertSub, BiLSTM on the IMDB dataset. By definition, CertSub and ASCC train for an arbitrary number of substitutions. CF Acc. is computed using ARC. Note that CertSub can only certify for $\{(T_{SubSyn}, \infty)\}$ and ASCC cannot certify.

ARC trains more robust models than CertSub for two perturbation spaces with word substitution. Table 7 shows that ARC achieves higher accuracy, CF Acc., and EX Acc. than CertSub on the two perturbation spaces.

ARC trains a more robust model than ASCC for $\{(T_{\mathbf{SubSyn}}, 1)\}$, but ASCC's model is more robust for $\{(T_{\mathbf{SubSyn}}, 2)\}$. Table 7 shows that the ARC-trained models have higher accuracy and CF Acc.

Comparison to POPQORN. We compare the certification of an ARC-trained model and a normal model against $\{(T_{SubSyn}, 3)\}$ on the first 100 examples in SST2 dataset. Because POPQORN can only certify the l_p norm ball, we overapproximate the radius of the ball as the maximum l_1 distance between the original word and its synonyms.

ARC runs much faster than POPQORN. ARC is more accurate than POPQORN on the ARC-trained model, while POPQORN is more accurate on the normal model. ARC certification takes 0.17sec/example on average for both models, while POPQORN certification takes 12.7min/example. ARC achieves 67% and 5% CF Acc. on ARC-trained model and normal model, respec-

Table 6: ARC vs A3T (CNN) on SST2 dataset.

		$\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$				$\{(T_{Dup}, 2), (T_{SubSyn}, 2)\}$			
Train	Model	Acc.	HF Acc.	CF Acc.	EX Acc.	Acc.	HF Acc.	CF Acc.	EX Acc.
A3T (HotFlip)	CNN	80.2	71.9	N/A	70.2	79.9	68.3	N/A	57.7
ARC	LSTM	**82.5**	**77.8**	**72.5**	**77.0**	**80.2**	**70.0**	**55.4**	**64.0**

Table 7: ARC vs CertSub and ASCC on IMDB dataset.

	$\{(T_{SubSyn}, 1)\}$			$\{(T_{SubSyn}, 2)\}$		
Train	Acc.	CF Acc.	EX Acc.	Acc.	CF Acc.	EX Acc.
CertSub	76.8	67.0	71.0	76.8	64.8	68.3
ACSS	82.8	0.0	81.5	82.8	0.0	**80.8**
ARC	**87.7**	**77.8**	**82.6**	**86.3**	**71.0**	78.2

tively, while POPQORN achieves 22% and 28% CF Acc., but crashes on 45% and 1% of examples for two models.

Comparison to SAFER (Ye et al., 2020). SAFER is a post-processing technique for certifying robustness via *randomized smoothing*. We train a Bi-LSTM model using ARC following SAFER's experimental setup on the IMDB dataset and SAFER's synonym set, which is different from CertSub's. We consider the perturbation spaces $\{(T_{SubSyn}, 1)\}$ and $\{(T_{SubSyn}, 2)\}$. We use both ARC and SAFER to certify the robustness. The significance level of SAFER is set to 1%.

SAFER has a higher certified accuracy than ARC. However, its certificates are statistical, tied to word substitution only, and are slower to compute. Considering $\{(T_{SubSyn}, 2)\}$, ARC results in a certified accuracy of 79.6 and SAFER results in a certified accuracy of 86.7 (see appendix). Note that certified accuracies are incomparable because SAFER computes certificates that only provide statistical guarantees. Also, note that ARC uses $O(n \prod_{i=1}^{n} \delta_i)$ forward passes for each sample, while SAFER needs to randomly sample thousands of times. In the future, it would be interesting to explore extensions of SAFER to ARC's rich perturbation spaces.

5.3 Effects of Perturbation Space Size

We fix the LSTM model (Table 4) trained using ARC on $\{(T_{Dup}, 2), (T_{SubSyn}, 2)\}$ and SST2. Then, we test this model's CF Acc. and HotFlip accuracy on $\{(T_{Dup}, \delta), (T_{SubSyn}, \delta)\}$ by varying δ from 1 to 5. We evaluate the EX Acc. only for $\delta = 1, 2, 3$, due to the combinatorial explosion.

ARC maintains a reasonable CF Acc. for increasingly larger spaces. Fig 4 shows the results, along with the maximum perturbation space size in the test set. ARC can certify 41.7% of the test set even when the perturbation space size grows to about 10^{10}. For $\delta = 1, 2, 3$, the CF Acc. is lower than the EX Acc. (8.2 on average), while the HF Acc. is higher than the EX Acc. (5.6 on average). Note that ARC uses a small amount of time to certify the entire test set, 3.6min on average, using a single V100 GPU, making it incredibly efficient compared to brute-force enumeration.

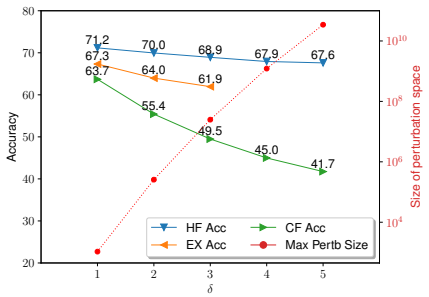


Figure 4: Accuracy metrics and the maximum perturbation space size as δ varies from 1 to 5 for $\{(T_{Dup}, \delta), (T_{SubSyn}, \delta)\}$.

6 Conclusion

We present ARC, which uses memoization and abstract interpretation to certify robustness to programmable perturbations for LSTMs. ARC can be used to train models that are more robust than those trained using existing techniques and handle more complex perturbation spaces. Last, the models trained with ARC have high certification accuracy, which can be certified using ARC itself.

Acknowledgements

We thank Nick Giannarakis and the anonymous reviewers for commenting on earlier drafts. This work is supported by the National Science Foundation grants CCF-1420866, CCF-1704117, CCF-1750965, CCF-1763871, CCF-1918211, CCF-1652140, the Microsoft Faculty Fellowship, and gifts and awards from Facebook.

References

Aws Albarghouthi. 2021. *Introduction to Neural Network Verification*. verifieddeeplearning.com. http://verifieddeeplearning.com.

Nicholas Carlini and David A. Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*, pages 3–14.

Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. 2019. Certified adversarial robustness via randomized smoothing. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320. PMLR.

Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages, POPL 1977, Los Angeles, California, USA, January 1977*, pages 238–252.

Xinshuai Dong, Anh Tuan Luu, Rongrong Ji, and Hong Liu. 2021. Towards robustness against natural language word substitutions. In *International Conference on Learning Representations*.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia. Association for Computational Linguistics.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 3–18.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Arthur Mann, and Pushmeet Kohli. 2019. Scalable verified training for provably robust image classification. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 4841–4850. IEEE.

Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. 2019. Achieving verified robustness to symbol substitutions via interval bound propagation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4083–4093, Hong Kong, China. Association for Computational Linguistics.

Yuval Jacoby, Clark W. Barrett, and Guy Katz. 2020. Verifying recurrent neural networks using invariant inference. In *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, pages 57–74.

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. 2019. Certified robustness to adversarial word substitutions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4129–4142, Hong Kong, China. Association for Computational Linguistics.

Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. 2019a. POPQORN: quantifying robustness of recurrent neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3468–3477. PMLR.

Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. 2019b. POPQORN: quantifying robustness of recurrent neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3468–3477. PMLR.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. Textbugger: Generating adversarial text against real-world applications. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*.

Linyi Li, Xiangyu Qi, Tao Xie, and Bo Li. 2020. Sok: Certified robustness for deep neural networks. *CoRR*, abs/2009.04131.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Paul Michel, Xian Li, Graham Neubig, and Juan Pino. 2019. On evaluation of adversarial perturbations for sequence-to-sequence models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3103–3114, Minneapolis, Minnesota. Association for Computational Linguistics.

Matthew Mirman, Timon Gehr, and Martin T. Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3575–3583. PMLR.

Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. 2020. Robustness verification for transformers. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Johannes Welbl, Po-Sen Huang, Robert Stanforth, Sven Gowal, Krishnamurthy (Dj) Dvijotham, Martin Szummer, and Pushmeet Kohli. 2020. Towards verified robustness under text deletion interventions. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic perturbation analysis for scalable certified robustness and beyond. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Mao Ye, Chengyue Gong, and Qiang Liu. 2020. SAFER: A structure-free approach for certified robustness to adversarial word substitutions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3465–3475, Online. Association for Computational Linguistics.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4944–4953.

Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. 2019. Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5564–5569, Florence, Italy. Association for Computational Linguistics.

Yuhao Zhang, Aws Albarghouthi, and Loris D'Antoni. 2020. Robustness to programmable string transformations via augmented abstract training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 11023–11032.

A Appendix

A.1 Proof of Lemmas and Theorems

Lemma 4.1

Proof. We prove Lemma 4.1 by induction on i, j and S.

Base case: $H^{\varnothing}_{0,0} = h_0$ is defined by both Eq 6 and Eq 8.

Inductive step for $H^S_{i,j}$: Suppose the lemma holds for $H^{S'}_{i',j'}$, where $(0 \leq i' \leq i \wedge 0 \leq j' \leq j \wedge S' \subseteq S) \wedge (i' \neq i \vee j' \neq j \vee S' \neq S)$. $S' \subseteq S$ denotes that for all $(T_k, \delta'_k) \in S'$, we have $(T_k, \delta_k) \in S$ and $\delta'_k \leq \delta_k$.

$H^S_{i,j}$ in Eq 6 comes from two cases illustrated in Fig 2. These two cases are captured by the first line and the second line in Eq 8, respectively. The inductive hypothesis shows that the lemma holds on states $H^S_{i-1,j-1}$ and $H^{S_{k\downarrow}}_{i-t_k,j-s_k}$. Thus, the lemma also holds on $H^S_{i,j}$. □

Theorem 4.1

Proof. We can expand Eq 5 using the decomposition of the perturbation space as

$$\begin{aligned} & F \\ &= \bigcup_{S' \in \text{DEC}_S} \{\text{LSTM}(\mathbf{z}, h_0) \mid \mathbf{z} \in S'^{=}(\mathbf{x})\} \\ &= \bigcup_{S' \in \text{DEC}_S} \bigcup_{i \geq 0} \{\text{LSTM}(\mathbf{z}, h_0) \mid \mathbf{z} \in S'^{=}(\mathbf{x}) \wedge \text{LEN}_{\mathbf{z}} = i\} \end{aligned} \tag{11}$$

Eq 11 and Eq 10 are equivalent, leading to the equivalence of Eq 5 and Eq 10. □

Theorem 4.2 Theorem 4.2

Proof. We first show that Eq 7 is equivalent to

$$\bigcup_{0 \leq i \leq \text{MAXLEN}_{\mathbf{x}}} H^S_{i,\text{LEN}_{\mathbf{x}}} \quad \text{, where}$$

$$\text{MAXLEN}_{\mathbf{x}} = \text{LEN}_{\mathbf{x}} + \sum_{(T_k,\delta_k) \in S} \max(t_k - s_k, 0)\delta_k$$

where $f_k : \Sigma^{s_k} \to 2^{\Sigma^{t_k}}$. As we will prove later, $\text{MAXLEN}_{\mathbf{x}}$ is the upper bound of the length of the perturbed strings. Because t_k, s_k, δ_k are typically small constants, we can regard $\text{MAXLEN}_{\mathbf{x}}$ as a term that is linear in the length of the original string $\text{LEN}_{\mathbf{x}}$, i.e., $\text{MAXLEN}_{\mathbf{x}} = O(\text{LEN}_{\mathbf{x}})$.

Now, we prove that $\text{MAXLEN}_{\mathbf{x}}$ is the upper bound of the length of the perturbed string. The upper bound $\text{MAXLEN}_{\mathbf{x}}$ can be achieved by applying all string transformations that increase the perturbed string's length and not applying any string transformations that decrease the perturbed string's length. Suppose a string transformation $T_k = (f_k, \varphi_k), f_k : \Sigma^{s_k} \to 2^{\Sigma^{t_k}}$ can be applied up to δ_k times, then we can apply it δ_k times to increase the perturbed string's length by $(t_k - s_k)\delta_k$.

The proof of soundness follows immediately from the fact that α, $\sqcup$, and $\widehat{\text{LSTM}}$ overapproximate their inputs, resulting in an overapproximation F.

The proof of complexity follows the property that the number of non-empty hyperrectangles $\widehat{H}^S_{i,j}$ is $O(\text{LEN}_{\mathbf{x}} \cdot \prod_{i=1}^n \delta_i)$. This property follows the definition of the string transformations and the tight perturbation space $S^{=}$. $\widehat{H}^S_{i,j}$ can be non-empty iff

$$i = j + \sum_{(T_k,\delta_k) \in S} (t_k - s_k)\delta_k, \quad \text{where } f_k : \Sigma^{s_k} \to 2^{\Sigma^{t_k}}$$

For each $\widehat{H}^S_{i,j}$, we need to enumerate through all transformations, so the complexity is $O(\text{LEN}_{\mathbf{x}} \cdot n \prod_{i=1}^n \delta_i)$ in terms of the number of LSTM cell evaluations. The interval bound propagation needs only two forward passes for computing the lower bound and upper bound of the hyperrectangles, so it only contributes constant time to complexity. In all, the total number of LSTM cell evaluations needed is $O(\text{LEN}_{\mathbf{x}} \cdot n \prod_{i=1}^n \delta_i)$. □

A.1.1 Comparison to Xu et al. (2020)

The dynamic programming approach proposed in Xu et al. (2020) is a special case of ARC where the perturbation space only contains substitutions. The abstract state $g_{i,j}$ in their paper (Page 6, Theorem 2) is equivalent to $\widehat{H}^{\{(T_{SubSyn},j)\}}_{i,i}$ in our paper.

A.2 Handling Attention

We have introduced the memoization and abstraction of final states in Section 4. Moreover, for LSTM architectures that compute attention of each state h_i, we would like to compute the interval abstraction of each state at the ith time step, denoted as $\widehat{H}_i$.

It is tempting to compute H_i as

$$H_i = \bigcup_{S' \in \text{DEC}} \bigcup_{0 \leq j \leq \text{LEN}_{\mathbf{x}}} H^{S'}_{i,j} \tag{12}$$

Unfortunately, Eq 12 does not contain states that are in the middle of a string transformation (see next example).

Example A.1. *Consider the string transformation T_{swap} that swaps two adjacent words and suppose $\mathbf{x}$ = "to the", $S = \{(T_{\text{swap}}, 1)\}$, then*

$H_1 = \{\text{LSTM}(\textit{“to”}, h_0), \text{LSTM}(\textit{“the”}, h_0)\}$. *However, the only non-empty state with* $i = 1$ *is* $H^{\varnothing}_{1,1} = \{\text{LSTM}(\textit{“to”}, h_0)\}$. *The state* $\text{LSTM}(\textit{“the”}, h_0)$ *is missing because it is in the middle of transformation* T_{swap}.

But Eq 12 is correct for $i = 2$ *because the transformation* T_{swap} *completes at time step 2.*

Think of the set of all strings in a perturbation space as a tree, like in Fig. 1(b), where strings that share prefixes share LSTM states. We want to characterize a subset $G^S_{i,j}$ of LSTM states at the ith layer where the perturbed prefixes have had *all* transformations in a space S applied on the original prefix $\mathbf{x}_{1:j}$ and are in the middle of transformation T_k. Intuitively, $G^S_{i,j}$ is a super set of $H^S_{i,j}$ defined in Section 4.

We formally define $G^S_{i,j}$ as follows:

$$G^S_{i,j} = \{\text{LSTM}(\mathbf{z}_{1:i}, h_0) \mid \mathbf{z} \in S^{=}(\mathbf{x}_{1:j}) \wedge L_{\mathbf{z}} \geq i\} \quad (13)$$

We rewrite Eq 13 by explicitly applying the transformations defining the perturbation space S, thus deriving our final equations:

$$G^S_{i,j} = \bigcup_{1 \leqslant k \leqslant n} \bigcup_{\substack{1 \leqslant l \leqslant t_j \\ \varphi_k(\mathbf{x}_{a:b}) = 1}} \left\{ \begin{matrix} \text{LSTM}(\mathbf{c}, h) \mid h \in G^{S_{k\downarrow}}_{i-l, j-s_k} \\ \mathbf{c} \in f_{k,:l}(\mathbf{x}_{a:b}) \end{matrix} \right\} \cup \{\text{LSTM}(x_j, h) \mid h \in G^S_{i-1,j-1}\} \quad (14)$$

where $a = j - s_k + 1$ and $b = j$. Notation $f_{k,:l}(\mathbf{x}_{a:b})$ collects the first l symbols for each $\mathbf{z}$ in $f_k(\mathbf{x}_{a:b})$, i.e.,

$$f_{k,:l}(\mathbf{x}_{a:b}) = \{\mathbf{z}_{1:l} \mid \mathbf{z} \in f_k(\mathbf{x}_{a:b})\}$$

$G^S_{i,j}$ contains (1) strings whose suffix is perturbed by $T_k = (\varphi_k, f_k)$ and the last symbol of $\mathbf{z}$ is the lth symbol of the output of T_k (the first line of Eq 14), and (2) strings whose suffix (the last character) is not perturbed by any transformations (the second line of Eq 14).

Then, H_i can be defined as

$$H_i = \bigcup_{S' \in \text{DEC}} \bigcup_{0 \leq j \leq \text{LEN}_{\mathbf{x}}} G^{S'}_{i,j}$$

Lemma A.1. *Eq 13 and Eq 14 are equivalent.*

The above lemma can be proved similarly to Lemma 4.1.

We use interval abstraction to abstract Eq 14 similarity as we did in Section 4.2. The total number of LSTM cell evaluation needed is $O(\text{LEN}_{\mathbf{x}} \cdot \max_{i=1}^{n}(t_i) \cdot n \prod_{i=1}^{n} \delta_i)$.

A.3 Handling Bi-LSTMs

Formally, We denote $\mathbf{x}^R$ as the reversed string $\mathbf{x}$. Suppose a transformation T has a match function φ and a replace function f, the reversed transformation $T^R = (\varphi^R, f^R)$ is defined as

$$\varphi^R(\mathbf{x}) = \varphi(\mathbf{x}^R), f^R(\mathbf{x}) = \{\mathbf{z}^R \mid \mathbf{z} \in f(\mathbf{x}^R)\} \quad \forall \mathbf{x} \in \Sigma^*$$

A.4 Handling Tree-LSTMs

Intuitively, we replace substrings in the formalization of LSTM with subtrees in the Tree-LSTM case. We denote the subtree rooted at node u as $\mathbf{t}_u$ and the size of $\mathbf{t}_u$ as $\text{SIZE}_{\mathbf{t}_u}$. The state H^S_u denotes the Tree-LSTM state that reads subtree $\mathbf{t}_u$ generated by a tight perturbation space S. The initial states are the states at leaf node u

$$H^{\varnothing}_u = \{\text{LSTM}(x_u, h_0)\}$$

and the final state is H^S_{root}.

We provide transition equations for three specific tree transformations Fig 3.

A.4.1 Merge states

For a non-leaf node v, we will merge two states, each from a child of v.

$$H^S_v = \bigcup_{S' \in \text{DEC}_S} \{\text{TRLSTM}(h, g) \mid h \in H^{S'}_{v_1} \wedge g \in H^{S-S'}_{v_2}\} \quad (15)$$

where v_1 and v_2 are children of v, and TRLSTM denotes the Tree-LSTM cell that takes two states as inputs. Notation $S - S'$ computes a tight perturbation space by subtracting S' from S. Formally, suppose

$$S = \{(T_1, \delta_1), (T_2, \delta_2), \ldots, (T_n, \delta_n)\}$$
$$S' = \{(T_1, \delta'_1), (T_2, \delta'_2), \ldots, (T_n, \delta'_n)\}$$

then

$$S - S' = \{(T_1, \delta_1 - \delta'_1), (T_2, \delta_2 - \delta'_1), \ldots, (T_n, \delta_n - \delta'_n)\}$$

Notice that Eq 15 is general to any tight perturbation space S containing these three tree transformations.

A.4.2 $T_{\textit{SubSyn}}$

We first show the computation of H^S_u for a leaf node u. The substitution only happens in the leaf nodes because only the leaf nodes correspond to words.

$$H^{\{T_{\textit{SubSyn}}, 1\}}_u = \{\text{LSTM}(c, h_0) \mid c \in f_{\textit{sub}}(x_u)\}$$

Table 8: String transformations for S_{review}.

Trans	Description
$T_{review1}$	substitute a phrase in the set A with another phrase in A.
$T_{review2}$	substitute a phrase in the set B with another phrase in B or substitute a phrase in C with another phrase in C.
$T_{review3}$	delete a phrase "one of" from "one of the most ..." or from "one of the ...est".
$T_{review4}$	duplicate a question mark "?" or an exclamation mark "!".

A.4.3 T_{Dup}

T_{Dup} can be seen as a subtree substitution at leaf node u.

$$H_u^{\{T_{Dup},1\}} = \{\text{TRLSTM}(h, h)\},$$

where $h = \text{LSTM}(x_u, h_0)$.

A.4.4 $T_{DelStop}$

Things get tricky for $T_{DelStop}$ because $\{(T_{DelStop}, \delta)\}$ can delete a whole subtree $\mathbf{t}_v$ if (1) the subtree only contains stop words and (2) $\text{SIZE}_{\mathbf{t}_v} \leq \delta$. We call such subtree $\mathbf{t}_u$ *deletable* if both (1) and (2) are true.

Besides the merging equation Eq 15, we provide another transition equation for H_v^S, where v is any non-leaf node with two children v_1, v_2 and a perturbation space $S = \{\ldots, (T_{DelStop}, \delta), \ldots\}$

$$H_v^S = \bigcup_{S' \in \text{DEC}_S} \{\text{TRLSTM}(h, g) \mid h \in H_{v_1}^{S'} \wedge g \in H_{v_2}^{S-S'}\} \cup \begin{cases} H_{v_2}^{S-S_{del}^{(1)}} & \text{(1) } \mathbf{t}_{v_1} \text{ is } \textit{deletable} \\ H_{v_1}^{S-S_{del}^{(2)}} & \text{(2) } \mathbf{t}_{v_2} \text{ is } \textit{deletable} \\ H_{v_2}^{S-S_{del}^{(1)}} \cup H_{v_1}^{S-S_{del}^{(2)}} & \text{both (1) and (2)} \\ \varnothing & \text{otherwise} \end{cases} \tag{16}$$

where

$$S_{del}^{(1)} = \{(T_{DelStop}, \text{SIZE}_{\mathbf{t}_{v_1}})\}, S_{del}^{(2)} = \{(T_{DelStop}, \text{SIZE}_{\mathbf{t}_{v_2}})\}$$

A.4.5 Soundness and Complexity

We use interval abstraction to abstract the transition equations for Tree-LSTM similarity as we did in Section 4.2. The total number of LSTM/Tree-LSTM cell evaluations needed is $O(\text{SIZE}_{\mathbf{t}}(\prod_{i=1}^{n} \delta_i)^2)$. The term $(\prod_{i=1}^{n} \delta_i)^2$ comes from Eq 15, as we need to enumerate S' for each S in the decomposition set.

A.5 Experimental Setup

We conduct all experiments on a server running Ubuntu 18.04.5 LTS with V100 32GB GPUs and Intel Xeon Gold 5115 CPUs running at 2.40GHz.

A.5.1 Definition for S_{review}

We design S_{review} by inspecting highly frequent n-grams in the movie review training set. Formally,

$$S_{review} = \{(T_{review1}, 1), (T_{review2}, 1), (T_{review3}, 1), (T_{review4}, 1), (T_{SubSyn}, 2)\}$$

where $T_{review1}$, $T_{review2}$, $T_{review3}$, and $T_{review4}$ are defined in Table 8 with

$$A = \{\text{"this is"}, \text{"this 's"}, \text{"it is"}, \text{"it 's"}\}$$
$$B = \{\text{"the movie"}, \text{"the film"}, \text{"this movie"}, \text{"this film"}, \text{"a movie"}, \text{"a film"}\}$$
$$C = \{\text{"the movies"}, \text{"the films"}, \text{"these movies"}, \text{"these films"}\}$$

A.5.2 Implementation of ARC

We provide a general implementation of ARC on LSTM against arbitrary user-defined string transformations. We also provide specific implementations of ARC on LSTM, Tree-LSTM, and Bi-LSTM against three transformations in Table 2 and Fig 3. Specific transformations allow us to optimize the specific implementations to utilize the full power of parallelism on GPU so that the specific implementations are faster than the general implementation. We conduct all our experiments on the specific implementations except for S_{review}.

A.5.3 Details of Training

A3T: A3T has two instantiations, A3T (HotFlip) and A3T (Enum). The difference between the two instantiations is the way it explores the augmentation space in A3T. We choose to show A3T (HotFlip) for comparison, but ARC wins over A3T (Enum) as well.

ASCC: ASCC updates the word embedding during training by defaults. In our experiments, we fix the word embedding for ASCC.

ARC: All the models trained by ARC have hidden state and cell state dimensions set to 100. We adopt a curriculum-based training method (Gowal et al., 2019; Huang et al., 2019; Jia et al., 2019;

Table 9: ARC vs SAFER on IMDB dataset.

	$\{(T_{SubSyn}, 1)\}$		$\{(T_{SubSyn}, 2)\}$	
Train	CF Acc.	RS Acc.	CF Acc.	RS Acc.
Data Aug.	0.2	**90.0**	0.1	**89.7**
ARC	**82.0**	87.2	**79.6**	86.7

Zhang et al., 2020) for training ARC by using a hyperparameter λ to weigh between the normal loss and the abstract loss and using a hyperparameter ϵ to gradually increasing the radius of synonym sets. We gradually change two hyperparameters from 0 to their maximum values by T_1 epochs and keep training with their maximum values by T_2 epochs.

Maximum values of hyperparameters λ and ϵ. For the experiments in Table 4, we tune the maximum of λ during training from 0.5 to 1.0 (with span 0.1) for LSTM and Bi-LSTM models and from 0.05 to 0.10 (with span 0.01) for Tree-LSTM models. For other experiments, which only use word substitutions, we fix the maximum of λ to be 0.8 following Jia et al. (2019).

For every experiment, the maximum of ϵ during training is defined by the size of word substitutions in the perturbation space. For example, $\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$ defines the maximum of ϵ as 2 and $\{(T_{DelStop}, 2), (T_{Dup}, 2)\}$ defines the maximum of ϵ as 0.

Epoch numbers T_1 and T_2. For LSTM and Bi-LSTM models on SST2 dataset, we set $T_1 = 16, T_2 = 8$. For other models (Tree-LSTM models on SST dataset and Bi-LSTM models on IMDB dataset), we set $T_1 = 20, T_2 = 10$.

We use early stopping for other training methods and step the early stopping epoch as 5.

We provide the training scripts and all trained models in supplementary materials.

A.6 Evaluation Results

The full results of comparison to SAFER are shown in Table 9.

Comparison to Huang et al. (2019). We use $\{(T_{SubSyn}, 3)\}$ on SST2 dataset for comparison between ARC and Huang et al. (2019). We directly quote the results in their paper.

ARC trains more robust LSTMs than CNNs trained by Huang et al. (2019). Table 10 shows that ARC results in models with higher accuracy (+1.6), HF Acc. (+1.1), CF Acc. (+28.8), and EX Acc. (+3.4) than those produced by Huang et al. (2019).

Table 10: ARC vs Huang et al. (2019) (CNN) on SST2 dataset.

		$\{(T_{SubSyn}, 3)\}$			
Train	Model	Acc.	HF Acc.	CF Acc.	EX Acc.
Huang et al.	CNN	81.7	77.2	44.5	76.5
ARC	LSTM	**83.3**	**78.3**	**73.3**	**77.9**

Table 11: Results of different instantiations of ARC-A3T on SST2 dataset.

	$\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$			$\{(T_{Dup}, 2), (T_{SubSyn}, 2)\}$		
Train	Acc.	CF Acc.	EX Acc.	Acc.	CF Acc.	EX Acc.
Abs-fir	**83.2**	15.1	68.6	82.6	15.9	**66.7**
Abs-sec	81.4	71.1	75.8	**83.0**	48.8	65.4
ARC	82.5	**72.5**	**77.0**	80.2	**55.4**	64.0

Effectiveness of ARC-A3T. We can apply the idea of A3T to ARC, extending ARC to abstract any subset of the given perturbation space and to augment the remaining perturbation space. We show the effectiveness of this extension in the appendix.

We evaluate ARC-A3T on the same perturbation spaces as we do for A3T. For each perturbation space, ARC-A3T has four instantiations: abstracting the whole perturbation space (downgraded to ARC), abstracting the first perturbation space ($\{(T_{DelStop}, 2)\}$ or $\{(T_{Dup}, 2)\}$), abstracting the second perturbation space ($\{(T_{SubSyn}, 2)\}$), and augmenting the whole perturbation space. We use enumeration for augmenting. We do not test the last instantiation because enumeration the whole perturbation space is infeasible for training. We further evaluate the trained models on different perturbation sizes, i.e., $\{(T_{DelStop}, \delta), (T_{SubSyn}, \delta)\}$ and $\{(T_{Dup}, \delta), (T_{SubSyn}, \delta)\}$ with $\delta = 1, 2, 3$.

Different instantiations of ARC-A3T win for different perturbation spaces. Table 11 shows the results of different instantiations of ARC-A3T. For $\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$, abstracting the first perturbation space ($\{(T_{DelStop}, 2)\}$) achieves the best accuracy and abstracting the whole perturbation space (ARC) achieves the best CF Acc. and EX Acc. For $\{(T_{Dup}, 2), (T_{SubSyn}, 2)\}$, abstracting the first perturbation space ($\{(T_{Dup}, 2)\}$) achieves the best EX Acc., abstracting the second perturbation space ($\{(T_{SubSyn}, 2)\}$) achieves the best accuracy, and abstracting the whole perturbation space (ARC) achieves the best CF Acc.

Figures 5 and 6 show the EX Acc. and CF Acc. for $\{(T_{DelStop}, \delta), (T_{SubSyn}, \delta)\}$ and $\{(T_{Dup}, \delta), (T_{SubSyn}, \delta)\}$, as δ varies from 1 to 3.

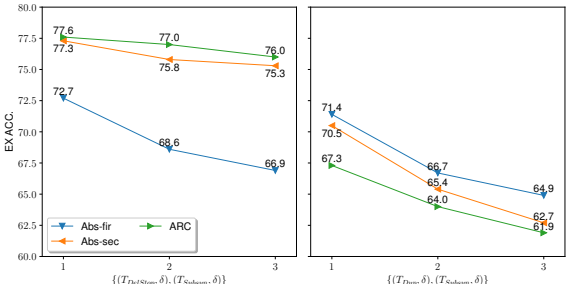


Figure 5: EX Acc. as δ varies from 1 to 3 for $\{(T_{DelStop}, \delta), (T_{SubSyn}, \delta)\}$ and $\{(T_{Dup}, \delta), (T_{SubSyn}, \delta)\}$.

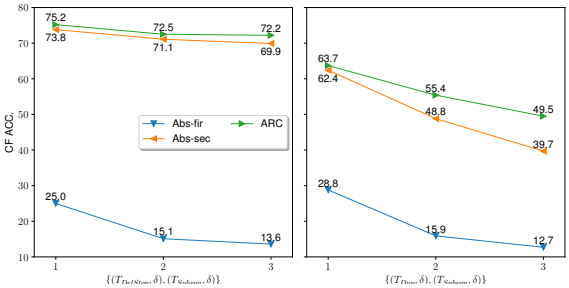


Figure 6: CF Acc. as δ varies from 1 to 3 for $\{(T_{DelStop}, \delta), (T_{SubSyn}, \delta)\}$ and $\{(T_{Dup}, \delta), (T_{SubSyn}, \delta)\}$.

ARC achieves the best EX Acc. and CF Acc. for $\{(T_{DelStop}, \delta), (T_{SubSyn}, \delta)\}$ and the best CF Acc. for $\{(T_{Dup}, \delta), (T_{SubSyn}, \delta)\}$. Abstracting the first perturbation space ($\{(T_{Dup}, \delta)\}$) achieves the best EX Acc. for $\{(T_{Dup}, \delta), (T_{SubSyn}, \delta)\}$. Notice that the abstraction approach proposed by ARC enables the abstracting of $\{(T_{Dup}, \delta)\}$.