⛵SAIL: Sound Abstract Interpreters with LLMs

Qiuhan Gu
UIUC

Avaljot Singh
UIUC

Gagandeep Singh
UIUC

Reasoning About Nonlinear Function

Q: Verify the property: $y \geq 0$?

```
1   def ReLU():
2       x = input()
3       if x < 0:
4           y = 0
5       else:
6           y = x
7       return y
```

Reasoning About Nonlinear Function

Q: Verify the property: $y \geq 0$?

```
1   def ReLU():
2       x = input()
3       if x < 0:
4           y = 0
5       else:
6           y = x
7       return y
```

- Enumerate all execution traces:

$x = -1 \rightarrow y = 1$ ✓

$x = 0 \rightarrow y = 0$ ✓

$x = 1 \rightarrow y = 1$ ✓

.... *Infinity!*

Reasoning About Nonlinear Function

Q: Verify the property: $y \geq 0$?

```
1   def ReLU():
2       x = input()
3       if x < 0:
4           y = 0
5       else:
6           y = x
7       return y
```

- Enumerate all execution traces:

$x = -1 \rightarrow y = 1$ ✓ $x = 0 \rightarrow y = 0$ ✓

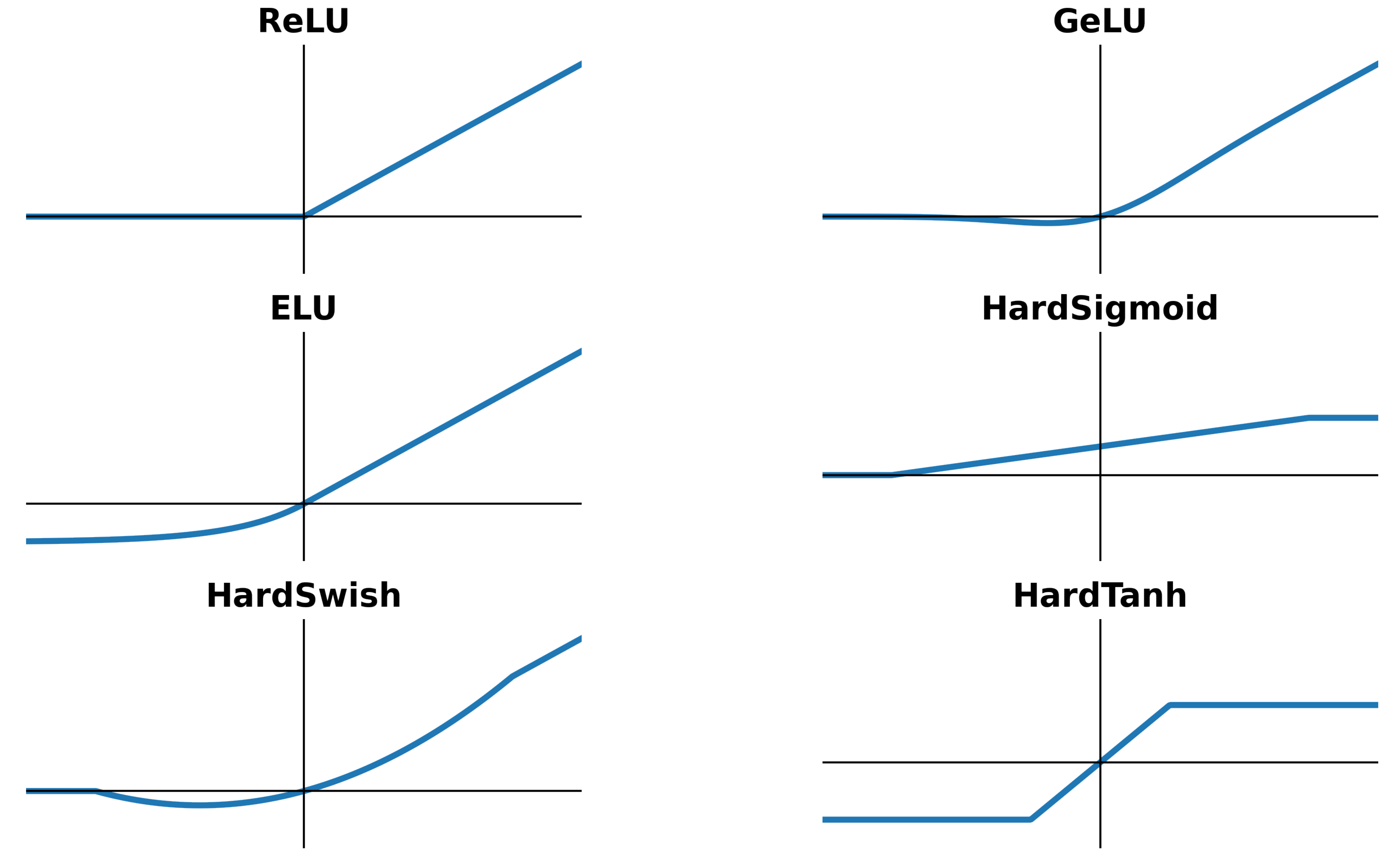
$x = 1 \rightarrow y = 1$ ✓ .... *Infinity!*

- Abstract the concrete input x:

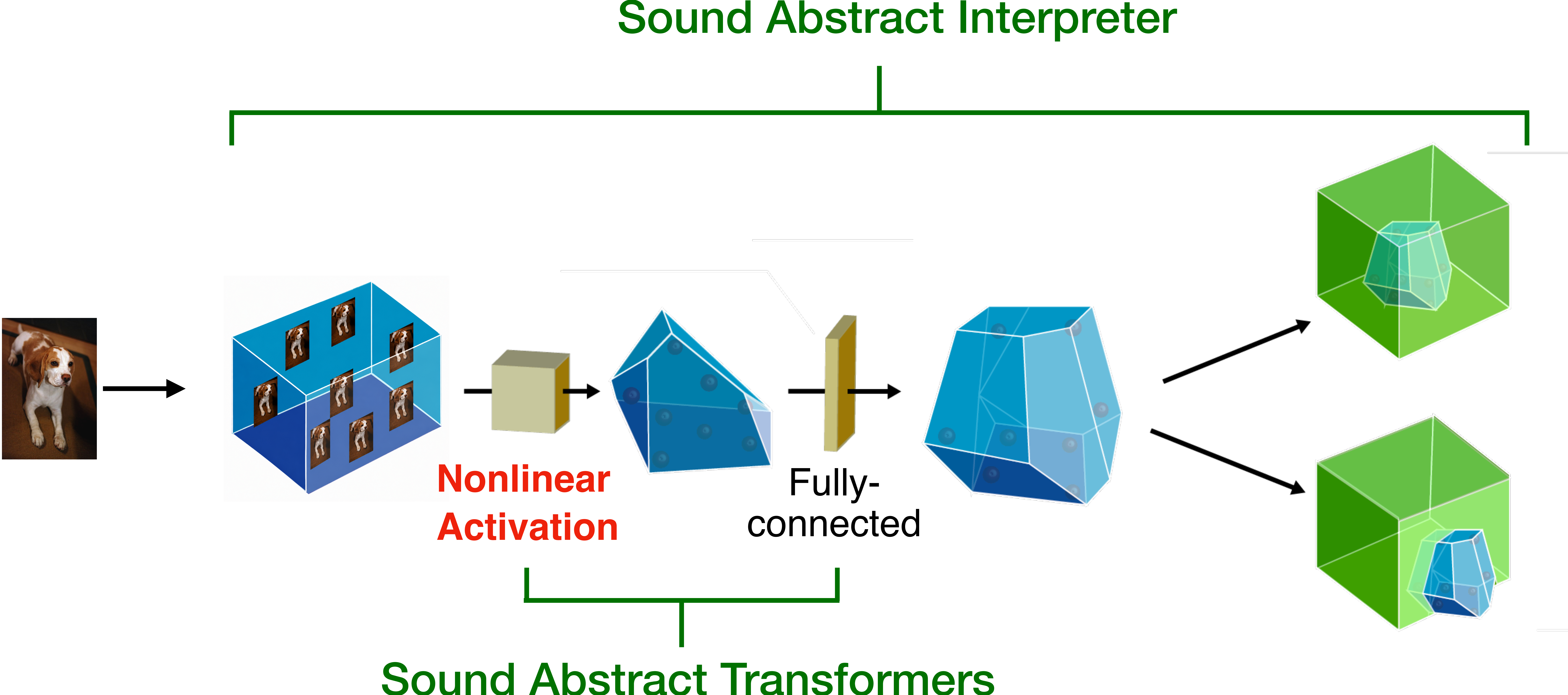
Case1: $x \in [0, +\infty) \rightarrow y \in [0, +\infty)$ ✓

Case2: $x \in (-\infty, 0] \rightarrow y \in [0, 0]$ ✓

Reasoning About Nonlinear Function



Example: Neural Network Verification



Constructing Sound Abstract Interpreter

- For each abstract transformer:

Manually Deriving Single Abstract Transformer

To clarify that our synthesis task is non-trivial, we provide more details about the representative example based on HardSigmoid in Section 3.1 of the paper. Specifically, let $h(x) = \mathrm{HardSigmoid}(x) = \min\{1, \max\{0, \frac{x}{6} + \frac{1}{2}\}\}$, and let the pre-activation interval be $x \in [l, u]$, $l, u \in \mathbb{R}$. A DeepPoly transformer returns a tuple $(\ell, u, L(x), U(x))$ such that for all $x \in [l, u]$, $\ell \le h(x) \le u \wedge L(x) \le h(x) \le U(x)$. We verify soundness by case analysis matching the synthesized transformer:

(1) **Case 1:** $u \le -3$.
The transformer outputs $(0, 0, 0, 0)$. Since $x \le u \le -3$, we have $\frac{x}{6} + \frac{1}{2} \le 0$, hence $h(x) = 0$ for all $x \in [l, u]$. Therefore $\ell = u = 0$ and $L(x) = U(x) = 0$ are sound.

(2) **Case 2:** $l \ge 3$.
The transformer outputs $(1, 1, 1, 1)$. Since $x \ge l \ge 3$, we have $\frac{x}{6} + \frac{1}{2} \ge 1$, hence $h(x) = 1$ for all $x \in [l, u]$. Therefore $\ell = u = 1$ and $L(x) = U(x) = 1$ are sound.

(3) **Case 3:** $-3 \le \ell \le u \le 3$.
The transformer outputs $\left(\frac{l}{6} + \frac{1}{2},\ \frac{u}{6} + \frac{1}{2},\ \frac{x}{6} + \frac{1}{2},\ \frac{x}{6} + \frac{1}{2}\right)$ over $[-3, 3]$. In this case $h(x) = \frac{x}{6} + \frac{1}{2}$, thus for all $x \in [l, u] \subseteq [-3, 3]$,

$$\ell = \min_{x \in [l,u]} h(x) = \frac{l}{6} + \frac{1}{2}, \qquad u = \max_{x \in [l,u]} h(x) = \frac{u}{6} + \frac{1}{2},$$

and $L(x) = U(x) = h(x)$ are sound.

(4) **Case 4:** $l \ge -3 \wedge u > 3$.
The transformer outputs $\left(\frac{l}{6} + \frac{1}{2},\ 1,\ L(x),\ 1\right)$, where $L(x) = ax + b$,

$$a = \frac{1 - \left(\frac{l}{6} + \frac{1}{2}\right)}{3 - l} = \frac{\frac{1}{2} - \frac{l}{6}}{3 - l}, \qquad b = \left(\frac{l}{6} + \frac{1}{2}\right) - al.$$

Equivalently, L is the secant line through $(l, \frac{l}{6} + \frac{1}{2})$ and $(3, 1)$.
To verify, we consider bounds separately.

- The interval bounds: since h is monotone non-decreasing, $\ell = h(l) = \frac{l}{6} + \frac{1}{2}$ and $u = 1$ are sound;
- The Upper affine bound: $U(x) = 1 \ge h(x)$ trivially;
- The lower affine bound: for $x \in [l, 3]$, we have $h(x) = \frac{x}{6} + \frac{1}{2}$, which is linear. $L(x)$ is the affine function that matches $h(x)$ at both endpoints $x = l$ and x coincides with $h(x)$ on the entire interval $[l, 3]$; for $x \in [3, u]$, $h(x) = 1$ and $L($ since L interpolates to 1 at $x = 3$ with non-positive slope beyond 3 in this case. $L(x) \le h(x)$ for all $x \in [l, u]$.

(5) **Case 5:** $l < -3 \wedge u \le 3$.
The transformer outputs $\left(0,\ \frac{u}{6} + \frac{1}{2},\ 0,\ U(x)\right)$, where $U(x) = cx + d$,

$$c = \frac{\frac{u}{6} + \frac{1}{2} - 0}{u - (-3)} = \frac{\frac{u}{6} + \frac{1}{2}}{u + 3}, \qquad d = -c(-3) = 3c.$$

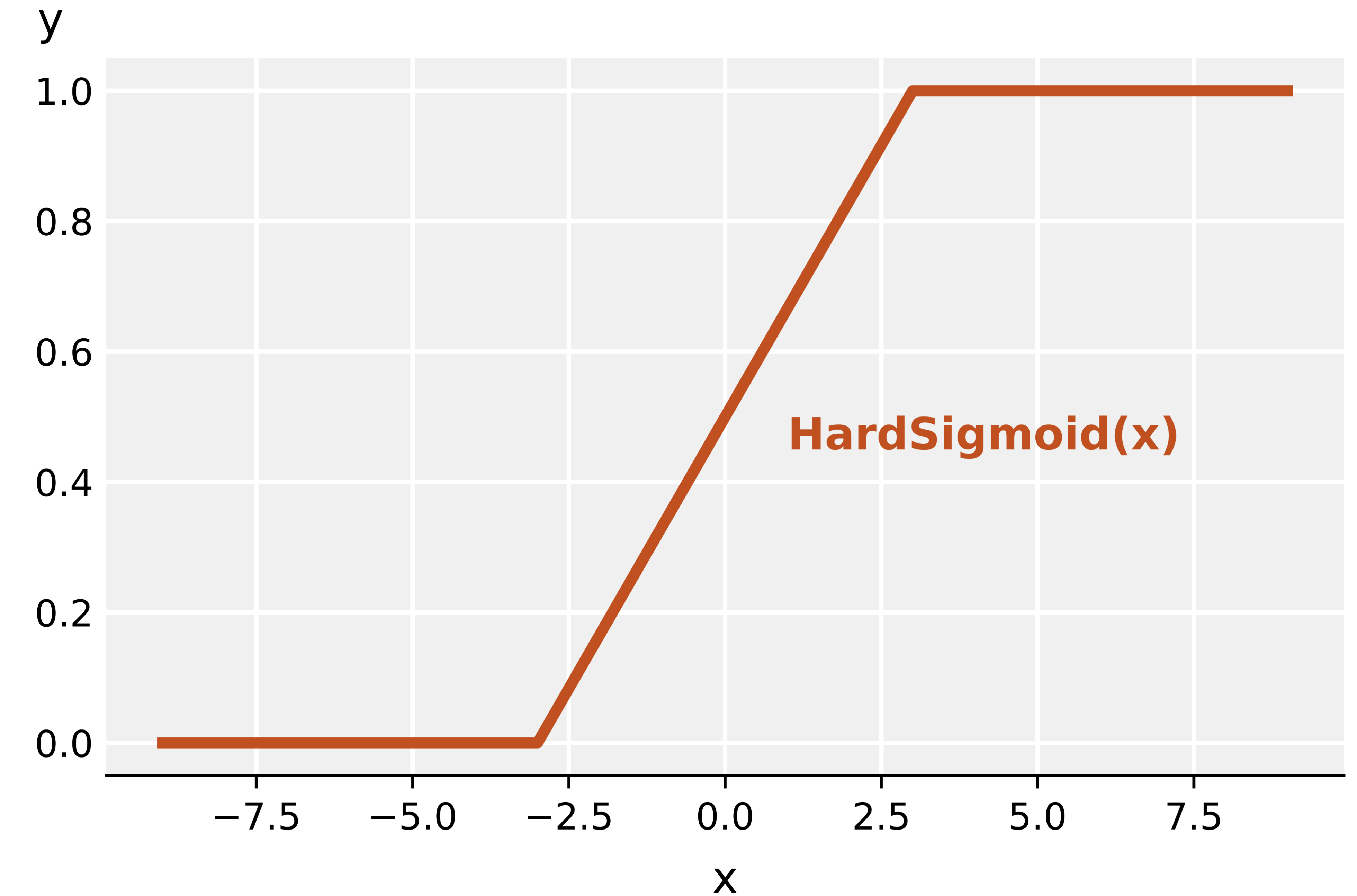
Equivalently, U is the secant line through $(-3, 0)$ and $(u, \frac{u}{6} + \frac{1}{2})$.
To verify, we consider bounds separately again.

- The interval bounds: monotonicity gives $\ell = 0$ and $u = h(u) = \frac{u}{6} + \frac{1}{2}$.
- The lower affine bound: $L(x) = 0 \le h(x)$ trivially.
- The upper affine: for $x \in [l, -3]$, $h(x) = 0$ and $U(x) \ge 0$ since $U(-3) = 0$, and U is non-decreasing on $[-3, u]$; for $x \in [-3, u]$, $h(x) = \frac{x}{6} + \frac{1}{2}$ and the secant line U is above this linear segment (equality at endpoints). Thus $h(x) \le U(x)$ for all $x \in [l, u]$.

(6) **Case 6 (otherwise):** $l < -3 \wedge u > 3$.
The transformer outputs $(0, 1, 0, 1)$. This is sound because $h(x) \in [0, 1]$ for all real x, and the constant affine bounds $L(x) = 0$ and $U(x) = 1$ satisfy $0 \le h(x) \le 1$ for all $x \in [l, u]$.

Therefore, if even the verification of a fixed candidate is complex, deriving such a transformer manually from scratch would be substantially more difficult.

However, SAIL can synthesize both the correct DSL and globally sound transforemers within decent time.



Automating Abstract Interpretation

Specification

- Symbolic Synthesis
- Optimization Methods

[1,2,3]

[1] Reps et al. (VMCAI'16): Automating Abstract Interpretation
[2] Kalita et al. (OOPSLA'22): Synthesizing Abstract Transformers.
[3] Paulsen et al. (TACAS'22): LinSyn: Synthesizing Tight Linear Bounds for Arbitrary Neural Network Activation Functions

Automating Abstract Interpretation

Specification → **Verification** → **Implementation**

- Symbolic Synthesis
- Optimization Methods

[1,2,3]

Constraintflow (SAS'24)

ProveSound (OOPSLA'25)

Constraintflow DSL

↓

Automatic Verifier with SMT Solving

Neuton (Arxiv'25)

Constraintflow DSL

↓

Pytorch

[1] Reps et al. (VMCAI'16): Automating Abstract Interpretation

[2] Kalita et al. (OOPSLA'22): Synthesizing Abstract Transformers.

[3] Paulsen et al. (TACAS'22): LinSyn: Synthesizing Tight Linear Bounds for Arbitrary Neural Network Activation Functions

Automating Abstract Interpretation

Specification → **Verification** → **Implementation**

Specification

- Symbolic Synthesis
- Optimization Methods

Constraintflow (SAS'24)

Verification

ProveSound (OOPSLA'25)

Constraintflow DSL

↓

Automatic Verifier with SMT Solving

Implementation

Neuton (Arxiv'25)

Constraintflow DSL

↓

Pytorch

Main Problems:

- ⚠️ Isolated Module
- ⚠️ Synthesis: Limited generality, Imprecision

The Problem

Can we have an end-to-end automatic pipeline?

SAIL: Sound Abstract Interpreters With LLMs

Can we have an end-to-end automatic pipeline?

SAIL: End-to-End Automation

Natural Language

SAIL

SAIL: End-to-End Automation

Natural Language

DSL

PyTorch

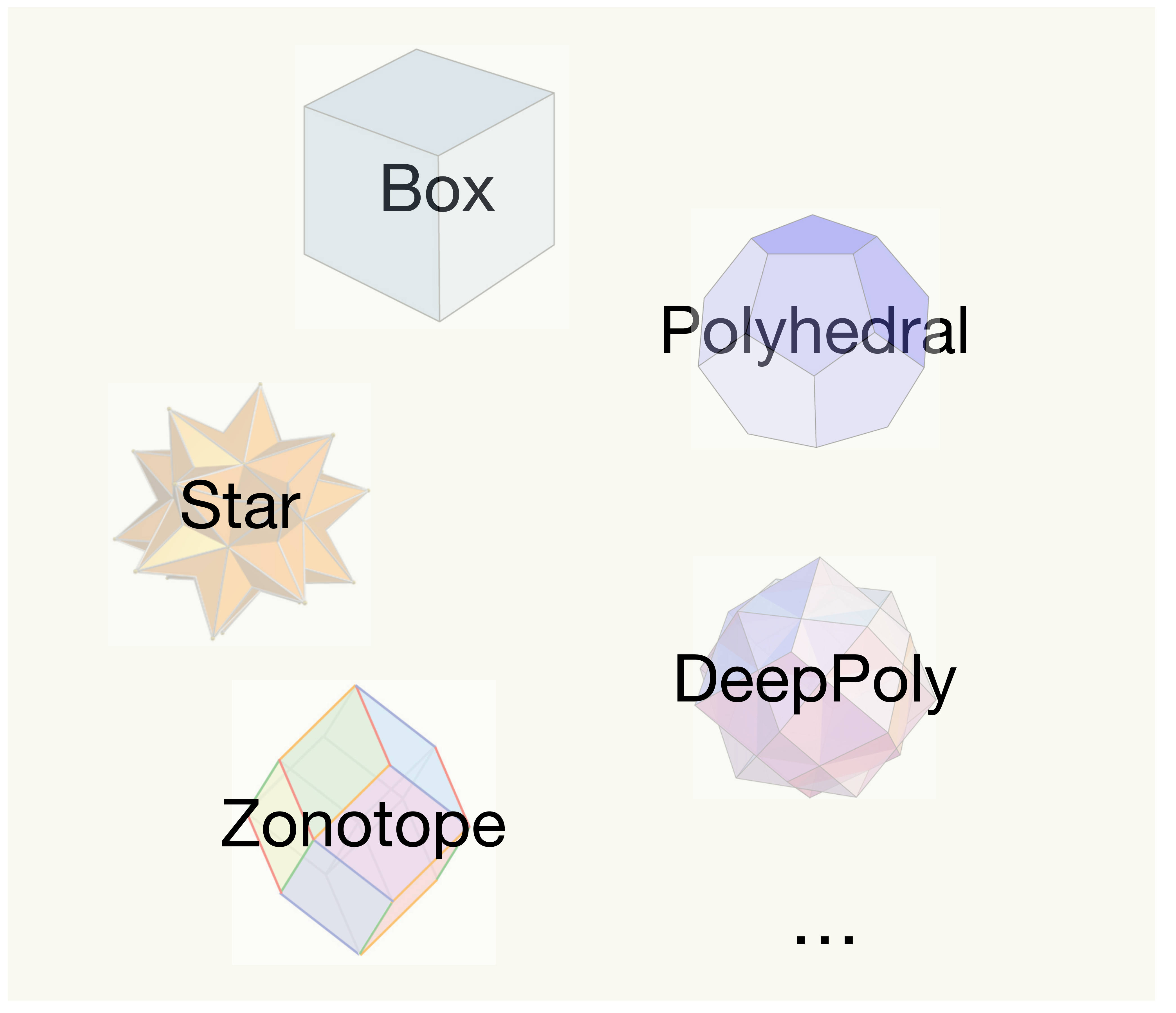
```
1  transformer DeepPoly {
2      HardSigmoid ->
3          (prev[u] <= -3) ?
4              (0, 0, 0, 0) :
5          (prev[l] >= 3) ?
6              (1, 1, 1, 1) :
7          (prev[l] >= -3) ?
8              (prev[u] <= 3) ?
9                  ((prev[l] / 6) + 0.5,
10                  (prev[u] / 6) + 0.5,
11                  (prev / 6) + 0.5,
12                  (prev / 6) + 0.5) :
13                 ((prev[l] / 6) + 0.5,
14                  1,
15                  (((0.5 - (prev[l] / 6)) / (prev[u] - prev[l])) * prev)
16                    + ((prev[l] / 6) + 0.5)
17                    - (((0.5 - (prev[l] / 6)) / (prev[u] - prev[l])) * prev[l]),
18                  1) :
19         (prev[u] <= 3) ?
20             (0,
21              (prev[u] / 6) + 0.5,
22              0,
23              (((((prev[u] / 6) + 0.5) / (prev[u] - prev[l])) * prev)
24                - ((((prev[u] / 6) + 0.5) / (prev[u] - prev[l])) * prev[l]))) :
25             (0, 1, 0, 1);
26 }
```

```
def HardSigmoid(self, abs_elem, prev, curr, poly_size, curr_size, prev_size, input_size, batch_size):
    cse_var_95 = SparseTensor([], [], 0, torch.tensor([]), dense_const=6, type= type(6)).unsqueeze(0).unsqueeze(1).repeat(
    cse_var_104 = abs_elem.get_elem('l', prev)
    cse_var_105 = SparseTensor([], [], 0, torch.tensor([]), dense_const=3, type= type(3)).unsqueeze(0).unsqueeze(1).repeat
    cse_var_91 = binary(binary(cse_var_104, cse_var_105, operator.add), cse_var_95, operator.truediv)
    cse_var_103 = SparseTensor([], [], 0, torch.tensor([]), dense_const=-3, type= type(-3)).unsqueeze(0).unsqueeze(1).repe
    cse_var_96 = binary(cse_var_104, cse_var_103, operator.ge)
    cse_var_106 = abs_elem.get_elem('u', prev)
    cse_var_97 = binary(cse_var_106, cse_var_105, operator.le)
    cse_var_324 = SparseTensor([], [], 0, torch.tensor([]), dense_const=1.0, type= type(1.0)).unsqueeze(0).unsqueeze(1).re
    cse_var_99 = binary(cse_var_104, cse_var_105, operator.ge)
    cse_var_100 = binary(cse_var_106, cse_var_103, operator.le)
    cse_var_102 = SparseTensor([], [], 0, torch.tensor([]), dense_const=0, type= type(0)).unsqueeze(0).unsqueeze(1).repeat
    cse_var_101 = where(cse_var_96, cse_var_91, cse_var_102)
    l_new = where(cse_var_100, cse_var_102, where(cse_var_99, cse_var_324, where(cse_var_97, cse_var_101, cse_var_101)))
    cse_var_94 = binary(binary(cse_var_106, cse_var_105, operator.add), cse_var_95, operator.truediv)
    rewrite_new_14 = convert_to_float(cse_var_96)
    u_new = where(cse_var_100, cse_var_102, where(cse_var_99, cse_var_324, where(cse_var_97, where(cse_var_96, cse_var_94,
    cse_var_93 = prev.convert_to_poly(abs_elem)
    cse_var_92 = SparseTensor([], [], 0, torch.tensor([]), dense_const=3, type= type(3)).unsqueeze(0).unsqueeze(1).repeat(
    cse_var_172 = cse_var_93.get_const()
    cse_var_173 = cse_var_93.get_mat(abs_elem)
    cse_var_254 = binary(cse_var_172, cse_var_92, operator.add)
    cse_var_75 = binary(cse_var_106, cse_var_104, operator.sub)
    cse_var_90 = binary(binary(cse_var_324, cse_var_91, operator.sub), cse_var_75, operator.truediv)
    cse_var_170 = repeat(cse_var_172, torch.tensor([batch_size, 1]))
    cse_var_171 = repeat(cse_var_173, torch.tensor([batch_size, 1, 1]))
    cse_var_256 = binary(cse_var_90, cse_var_170, operator.mul)
    cse_var_257 = binary(repeat(cse_var_90.unsqueeze(2), torch.tensor([1, 1, poly_size])), cse_var_171, operator.mul)
    cse_var_79 = PolyExpSparse(abs_elem.network, 0.0, cse_var_324)
    cse_var_63 = cse_var_79.get_const()
    cse_var_70 = repeat(cse_var_97.unsqueeze(2), torch.tensor([1, 1, poly_size]))
    cse_var_71 = cse_var_79.get_mat(abs_elem)
```

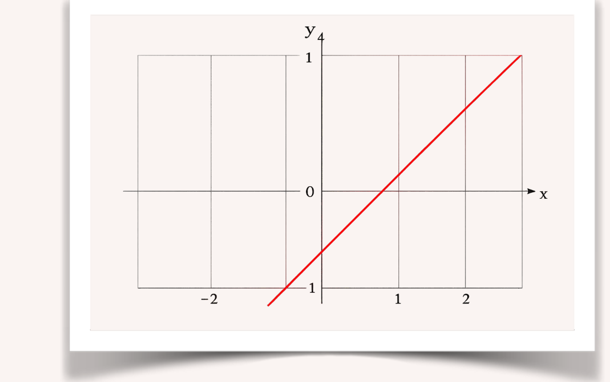
```
    cse_var_240 = binary(repeat(cse_var_74.unsqueeze(2), torch.tensor([1, 1, poly_size])), cse_var_171, operator.mul)
    cse_var_66 = PolyExpSparse(abs_elem.network, 0.0, SparseTensor([], [], 0, torch.tensor([]), dense_const=1, type= type(
    cse_var_234 = where(cse_var_96, cse_var_249, repeat(cse_var_66.get_const(), torch.tensor([batch_size, 1])))
    cse_var_236 = where(cse_var_85, cse_var_251, repeat(cse_var_66.get_mat(abs_elem), torch.tensor([batch_size, 1, 1])))
    cse_var_238 = binary(cse_var_239, binary(cse_var_74, cse_var_104, operator.mul), operator.sub)
    cse_var_235 = where(cse_var_96, cse_var_249, cse_var_238)
    cse_var_237 = where(cse_var_85,
    cse_var_232 = where(cse_var_97,   (variable) cse_var_237: SparseTensor
    cse_var_233 = where(cse_var_70, cse_var_237, cse_var_236)
    cse_var_230 = where(cse_var_99, cse_var_63, cse_var_232)
    cse_var_231 = where(cse_var_72, cse_var_71, cse_var_233)
    U_new = PolyExpSparse(abs_elem.network, where(cse_var_73, cse_var_84, cse_var_231) , where(cse_var_100, cse_var_77, cs
    return l_new, u_new, L_new, U_new
```

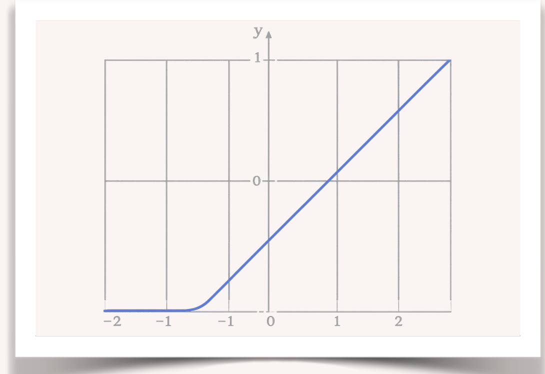
SAIL: One-Size-Fit-All Framework

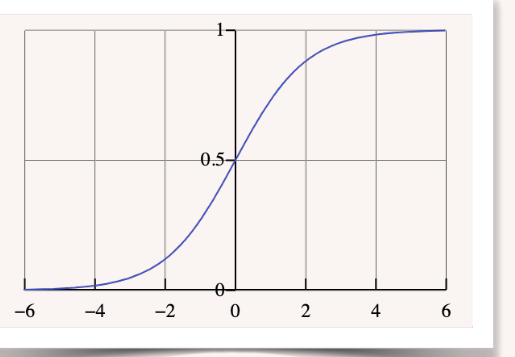
Multiple domains x **Multiple** transformers



- Linear
- Piecewise-linear
- Nonlinear







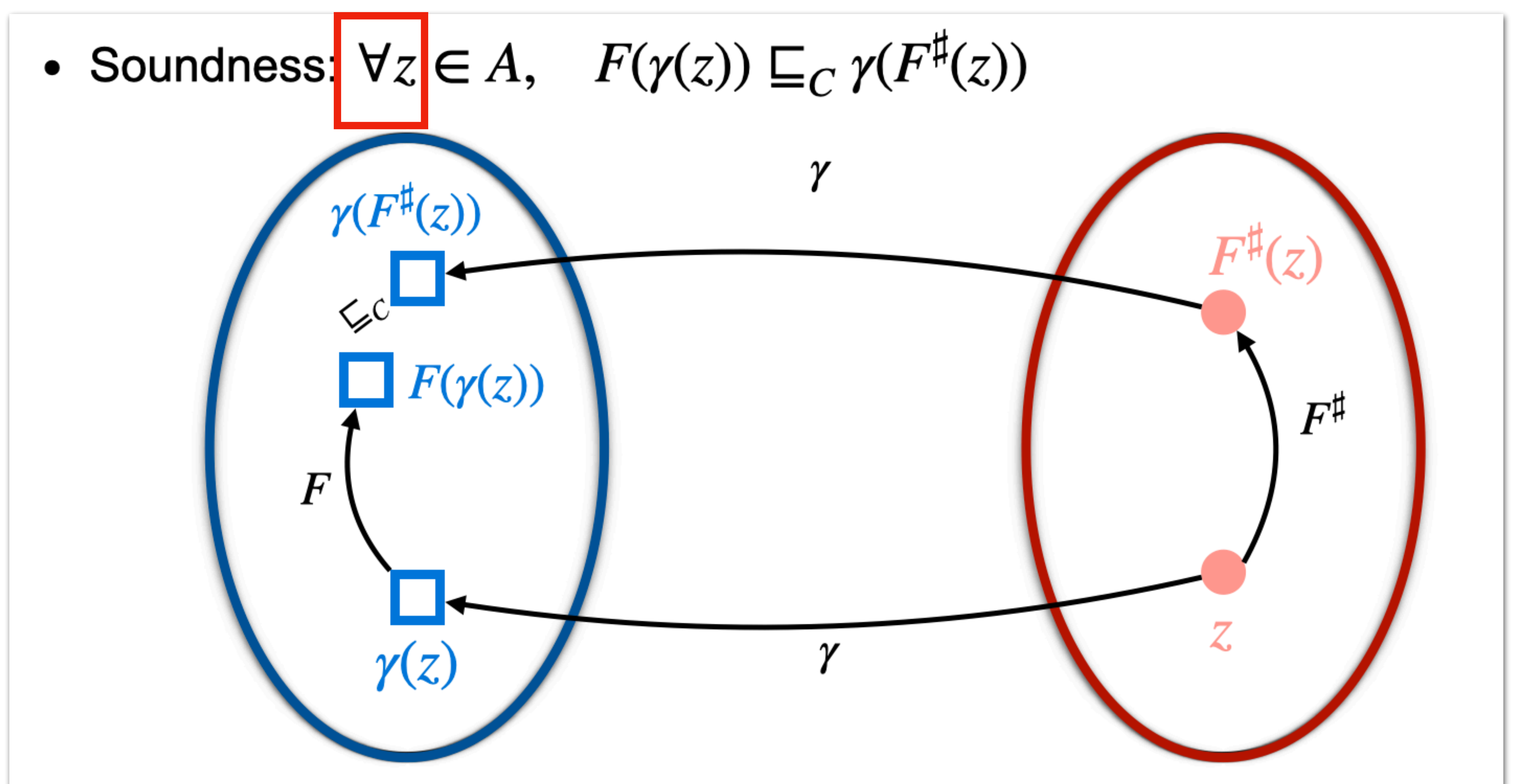
…

LLMs: Why Naïve Prompt Engineering Fails

LLMs: Why Naïve Prompt Engineering Fails

We require strict soundness guarantee.

1. Soundness Guarantee



- Soundness must hold for **all** abstract elements, which form an *infinite* space.
- Each abstract element may abstract an *infinite* number of concrete points.

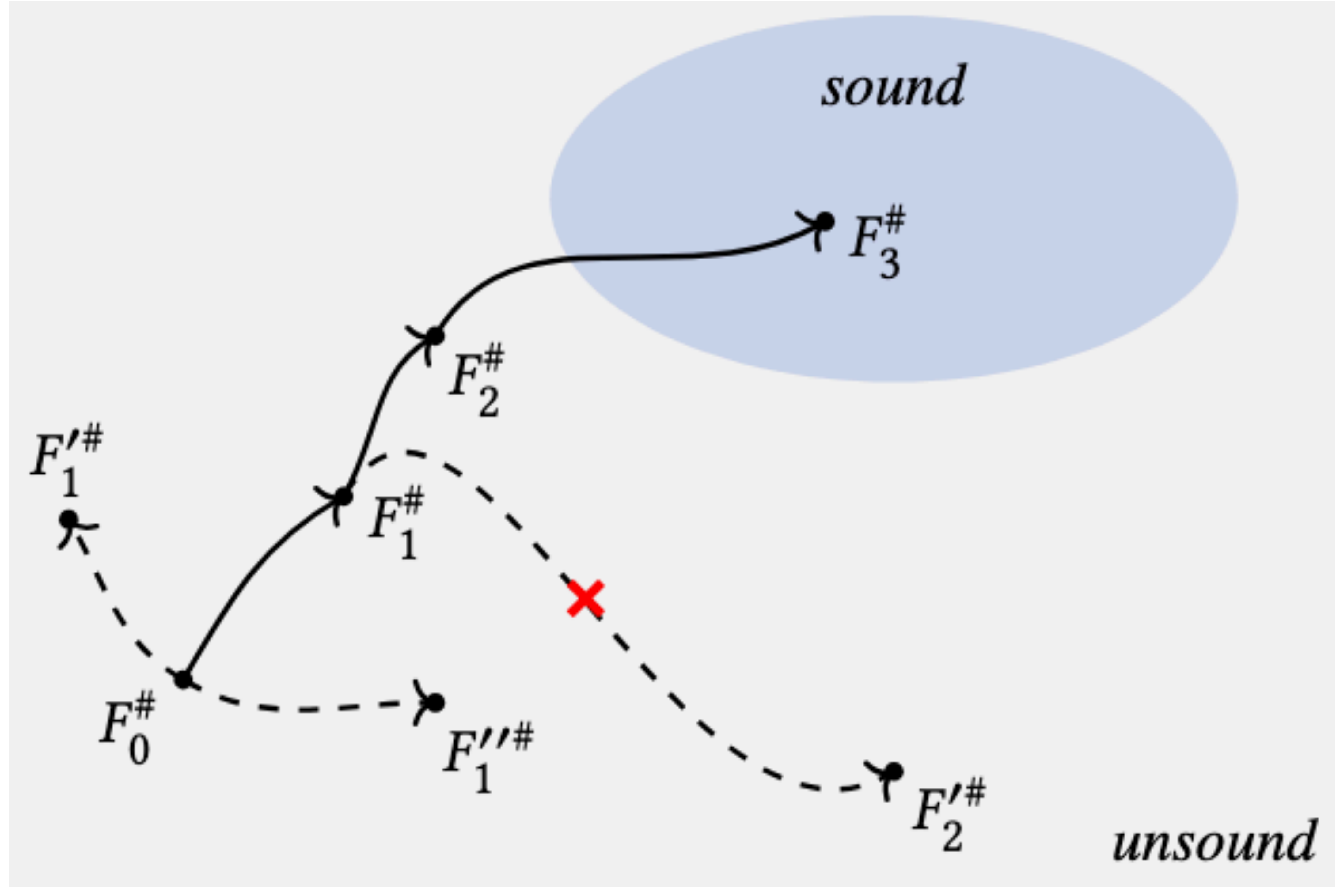
A Novel Principled Cost Function

Problem: How to guide LLMs in an infinite search space?

A function to measure the unsoundness

A Novel Principled Cost Function

Search blindly → Cost Function → **Measure** how far an unsound transformer deviates from being sound



LLMs: Why Naïve Prompt Engineering Fails

We need more than mathematical expressions.

2. Validity Guarantee

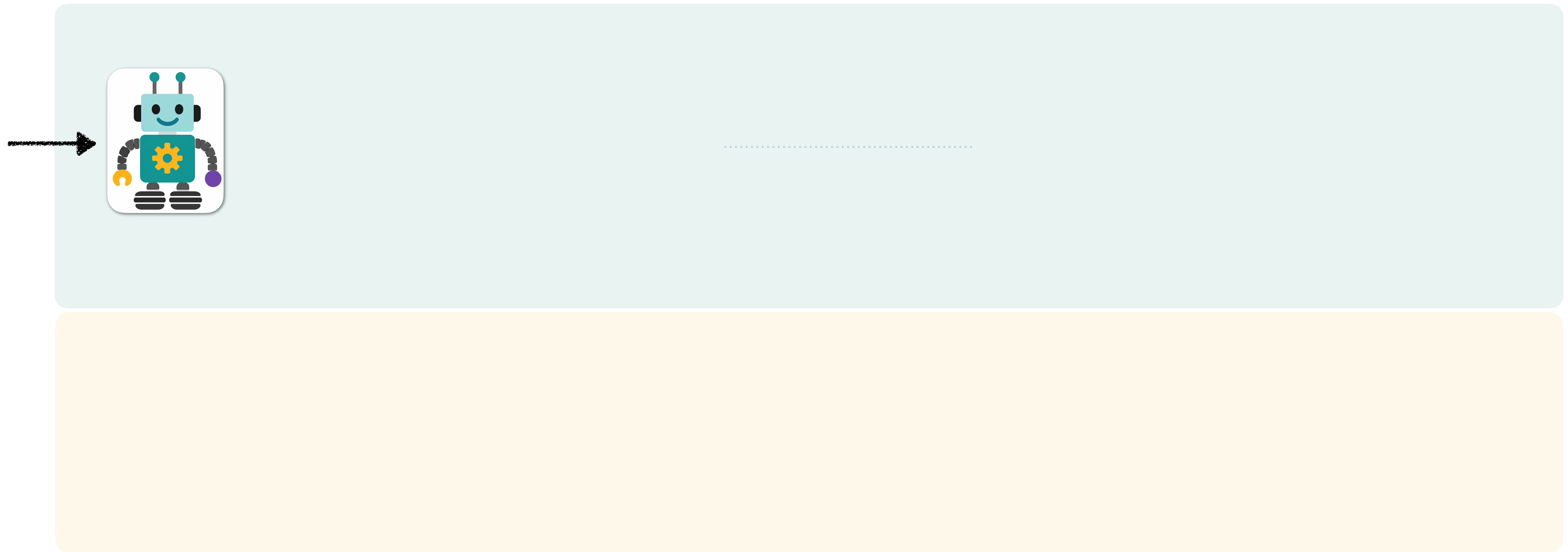
⟨*Expression*⟩	$e ::= c \mid x \mid \texttt{sym} \mid e_1 \oplus e_2 \mid e[x] \mid f_c(e_1, \cdots) \mid x.\texttt{traverse}(\delta, f_{c_1}, f_{c_2}, f_{c_3})\{e\} \mid e.\texttt{map}(f_c) \mid \texttt{solver}(\texttt{minimize}, e_1, e_2) \mid \cdots$
⟨*Shape-decl*⟩	$d ::= \texttt{Def shape as } (t_1\ x_1, t_2\ x_2, \cdots)\{e\}$
⟨*Function-def*⟩	$f ::= \texttt{Func } x(t_1\ x_1, t_2\ x_2, \cdots) = e$
⟨*DNN-operation*⟩	$\eta ::= \texttt{Affine} \mid \texttt{ReLU} \mid \texttt{MaxPool} \mid \texttt{DotProduct} \mid \texttt{Sigmoid} \mid \texttt{Tanh} \mid \cdots$
⟨*Transformer-decl*⟩	$\theta_d ::= \texttt{Transformer } x$
⟨*Transformer-ret*⟩	$\theta_r ::= (e_1, e_2, \cdots) \mid (e\ ?\ \theta_{r_1} : \theta_{r_2})$
⟨*Transformer*⟩	$\theta ::= \theta_d\ \{\eta_1 \rightarrow \theta_{r_1}; \eta_2 \rightarrow \theta_{r_2}; \cdots\}$
⟨*Statement*⟩	$s ::= \texttt{Flow}(\delta, f_{c_1}, f_{c_2}, \theta_c) \mid f \mid \theta \mid s_1\ ;\ s_2$
⟨*Program*⟩	$\Pi ::= d\ ;\ s$

Constrained Optimization Problem

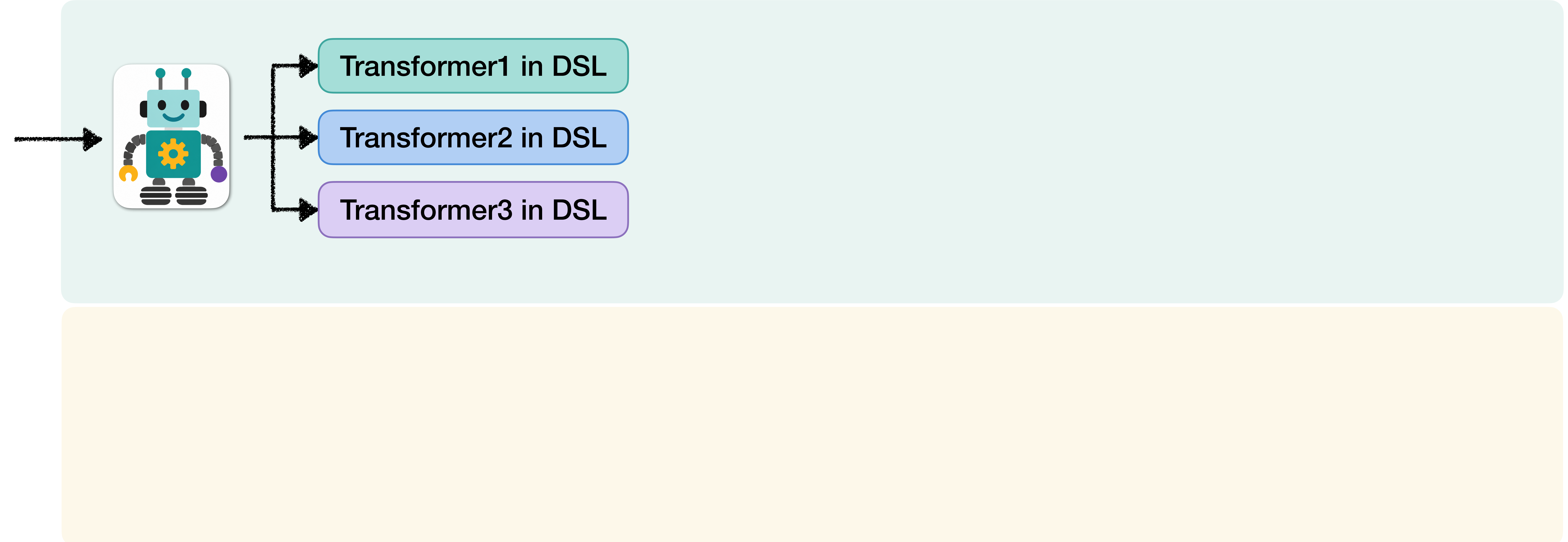
💡 **Idea #2:**

$F^{\sharp *} = \arg\min_{F^\sharp} L(F^\sharp)$ s.t., $F^\sharp$ satisfies all syntactic and semantic validity constraints.

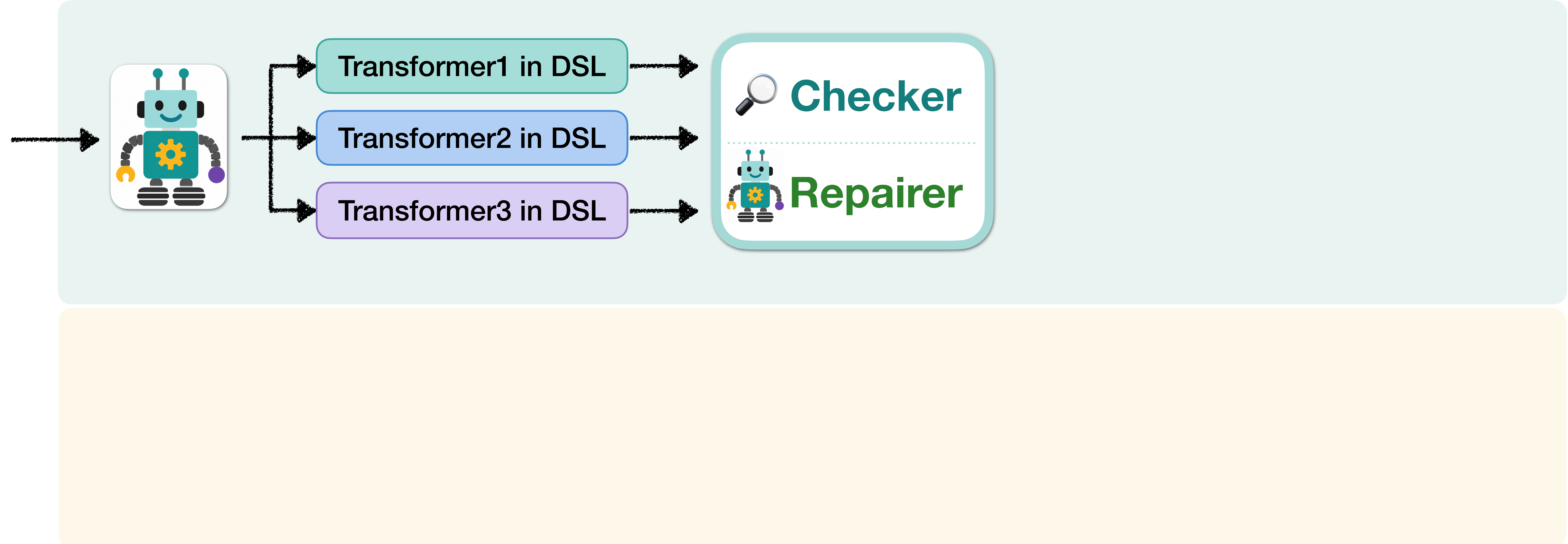
Constrained Optimization Problem



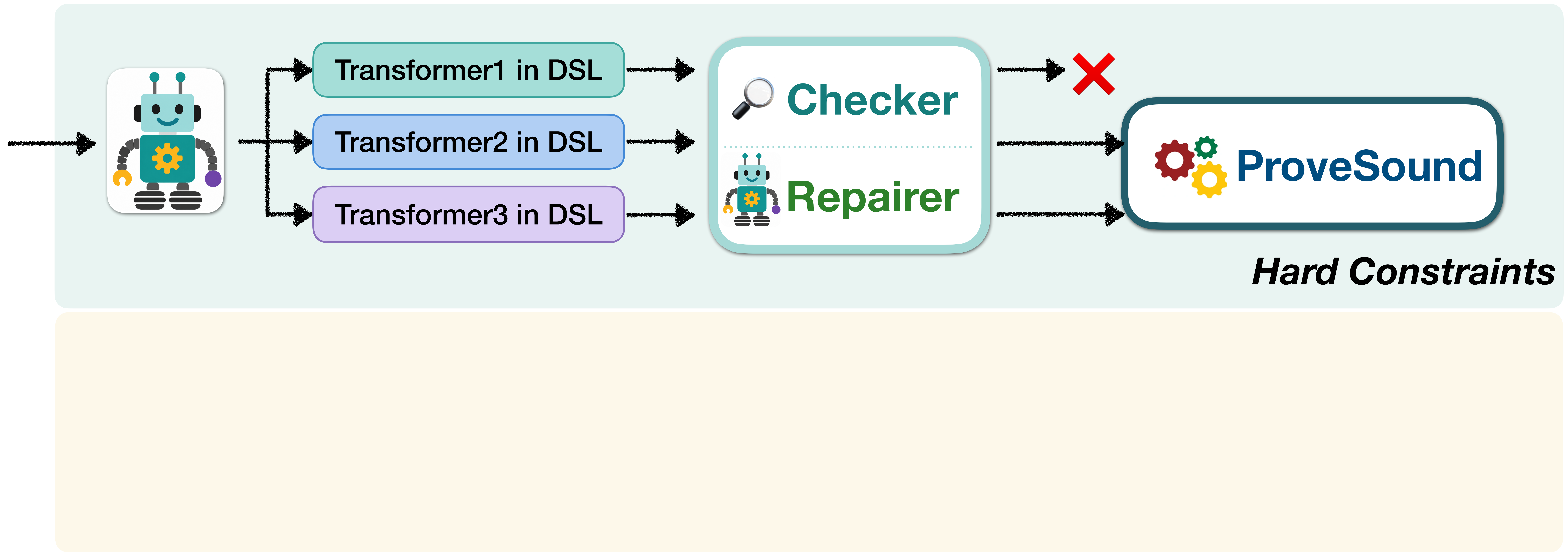
Constrained Optimization Problem



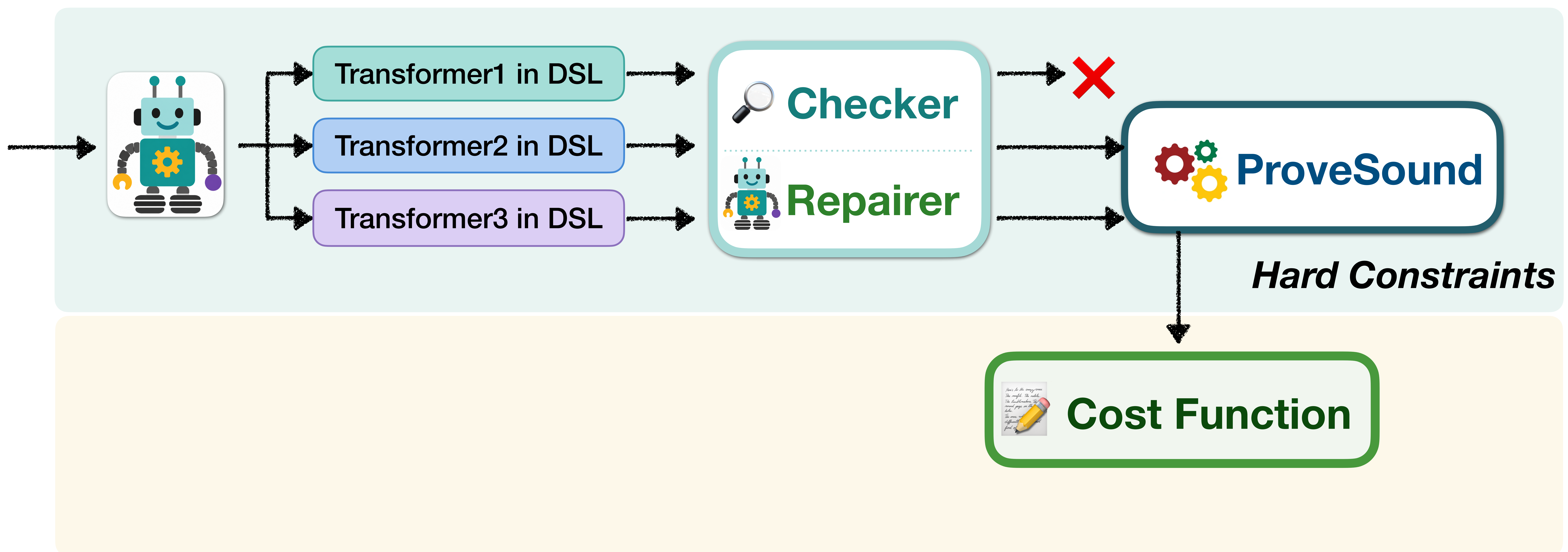
Constrained Optimization Problem



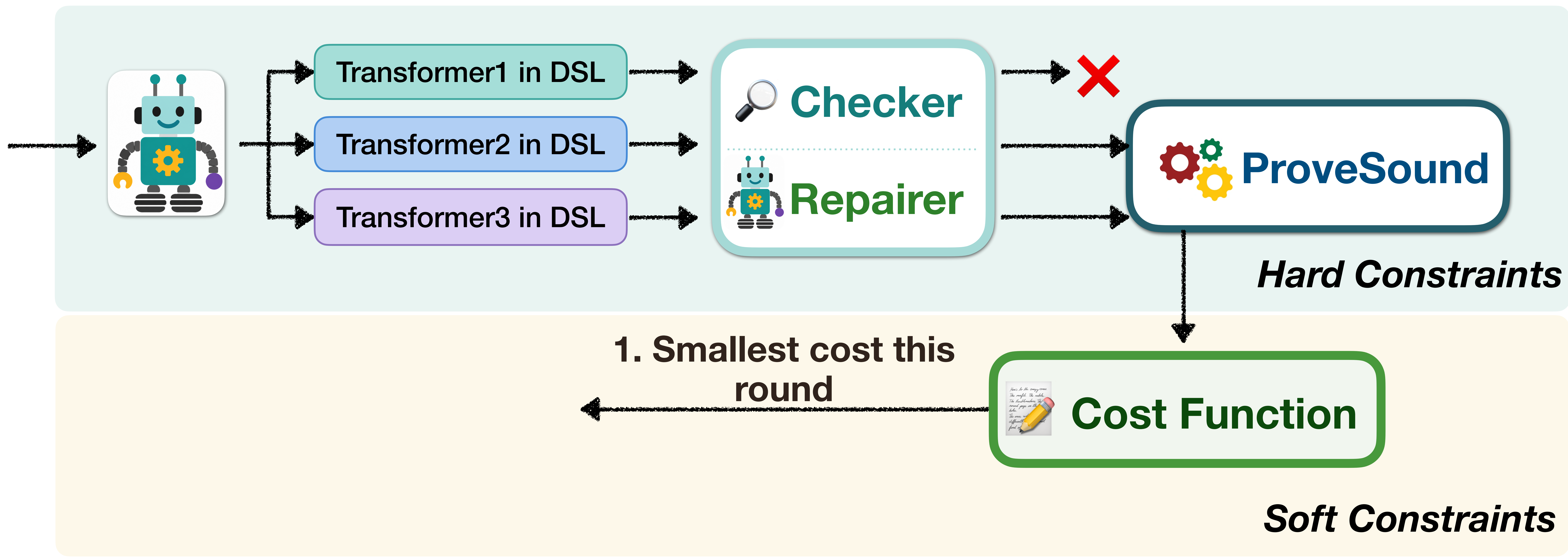
Constrained Optimization Problem



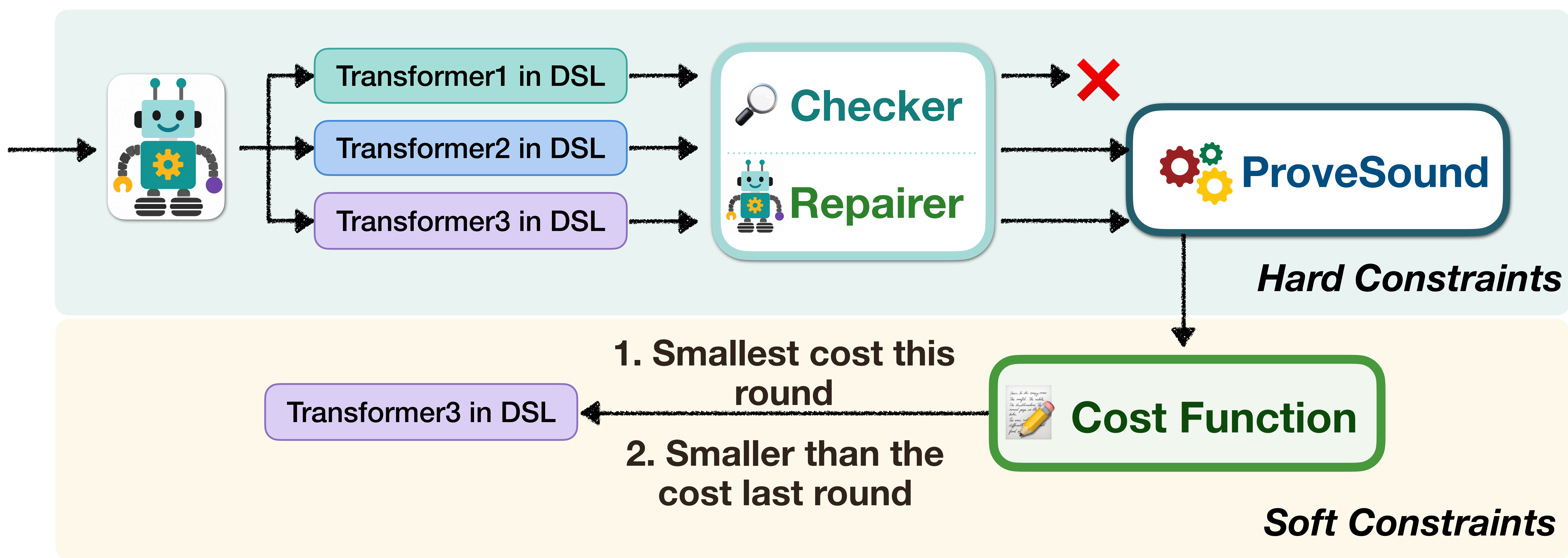
Constrained Optimization Problem



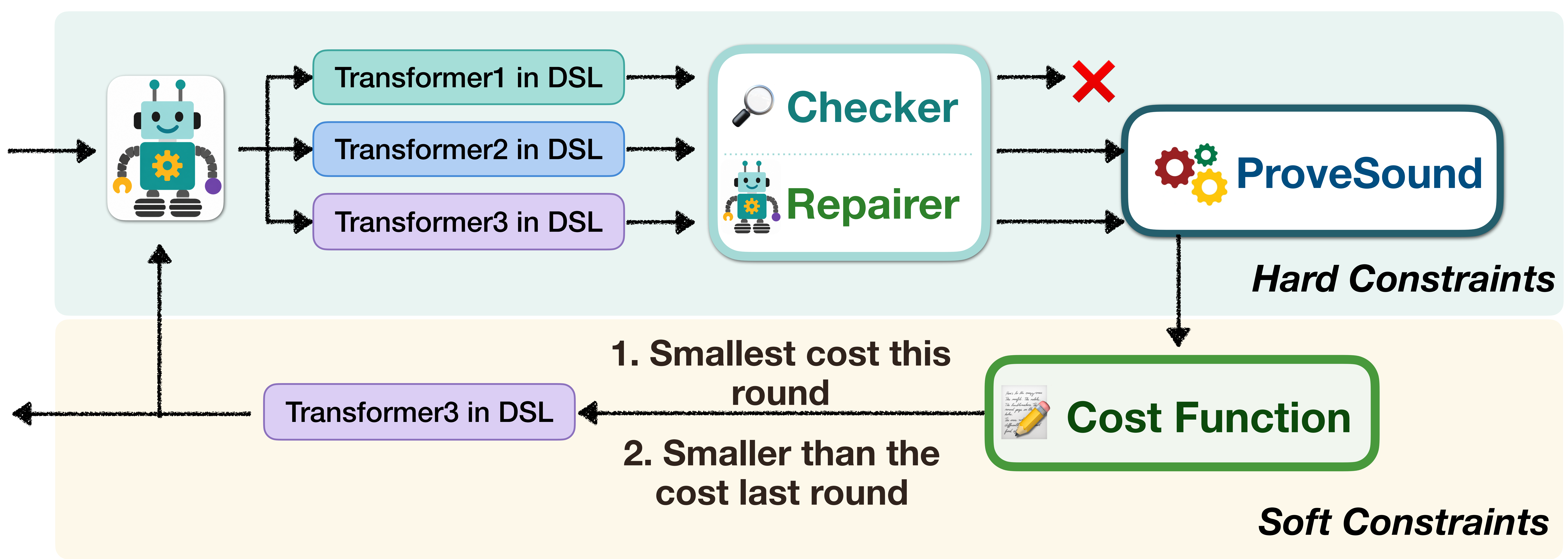
Constrained Optimization Problem



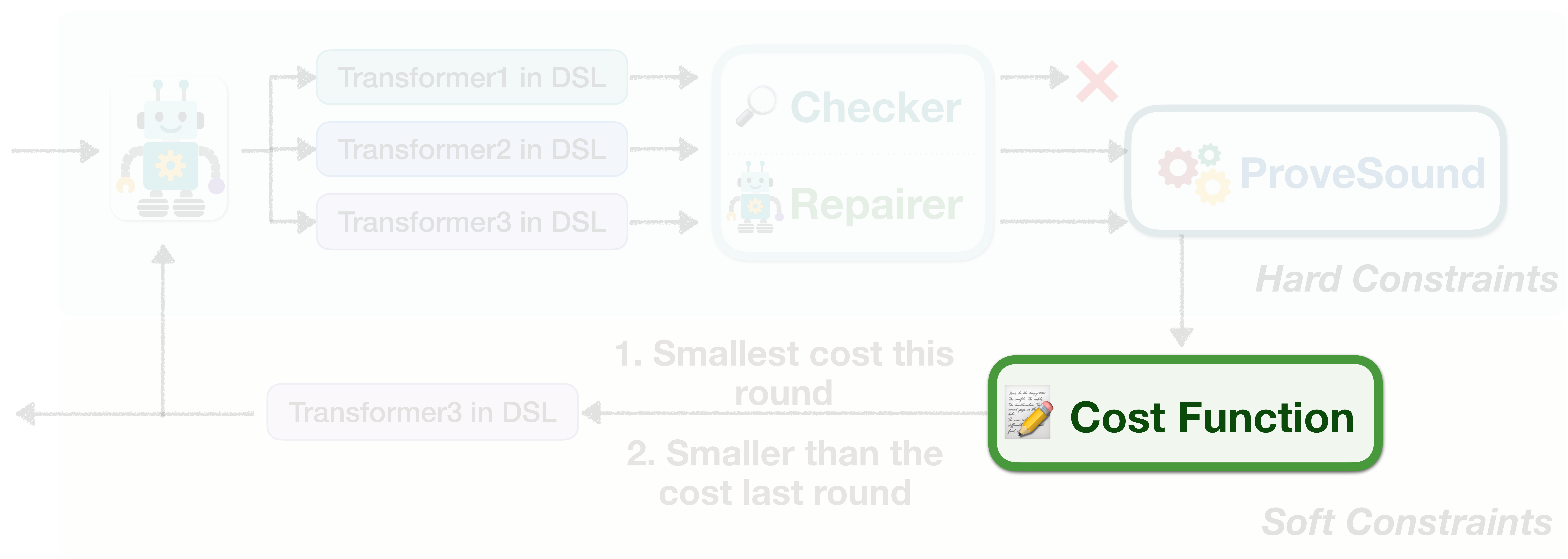
Constrained Optimization Problem



Constrained Optimization Problem



Constrained Optimization Problem



How to Design the Cost Function?

Example: $ReLU(x) = max(0, x)$, Box domain

How to Design the Cost Function?

Example: $ReLU(x) = max(0,\ x)$, Box domain

Sound Transformer: $ReLU^{\sharp}([l, u]) = [-\infty, \infty];$

$ReLU^{\sharp}([l, u]) = [0,\ max(0,\ u)];$

$\cdots$

How to Design the Cost Function?

Example: $ReLU(x) = max(0,\ x)$, Box domain

Sound Transformer: $ReLU^\sharp([l, u]) = [-\infty, \infty]$;

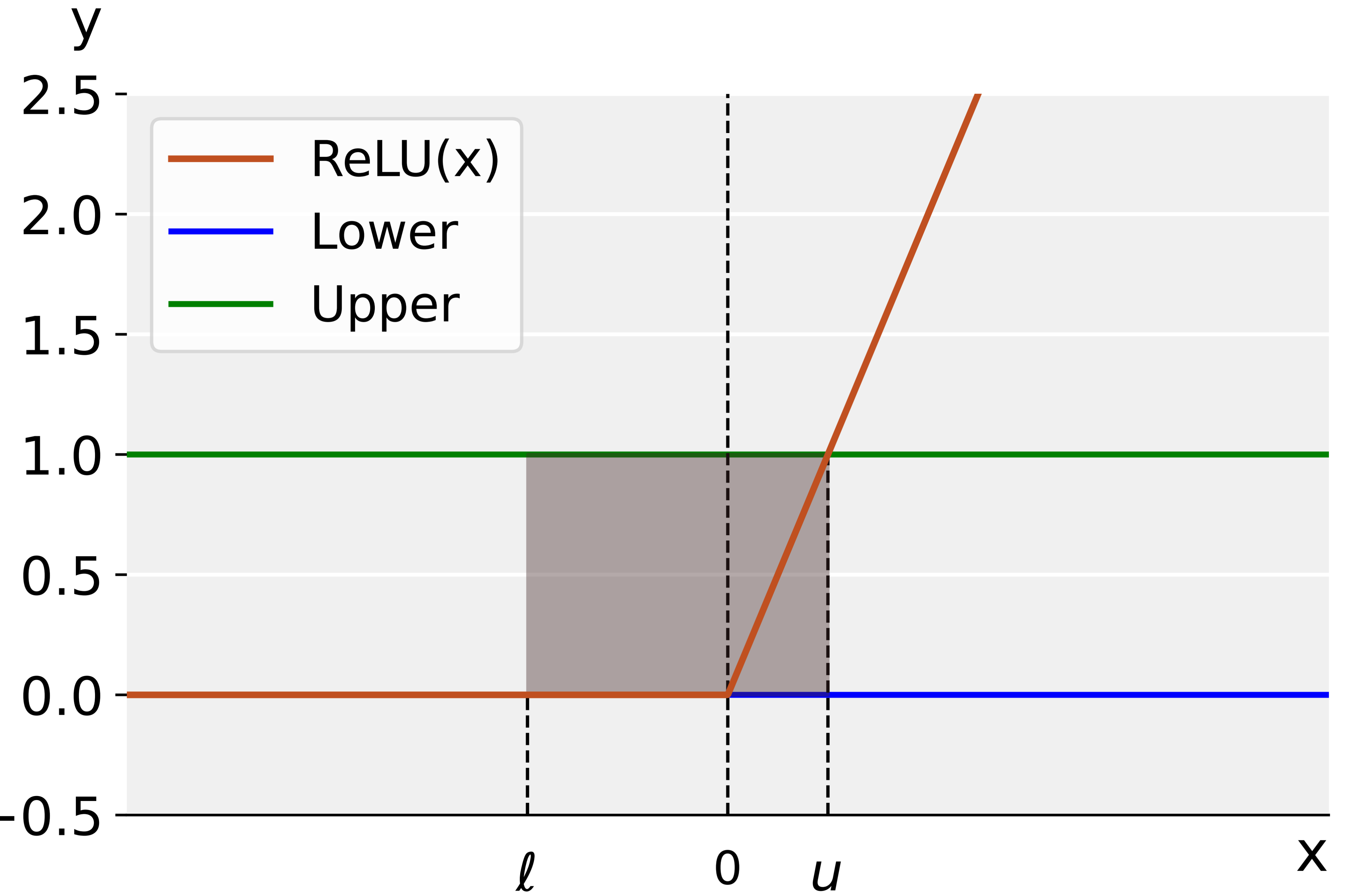
$ReLU^\sharp([l, u]) = [0,\ max(0,\ u)]$;

…

Unsound Transformer: $ReLU^\sharp([l, u]) = [0,1]$

Learning from Examples

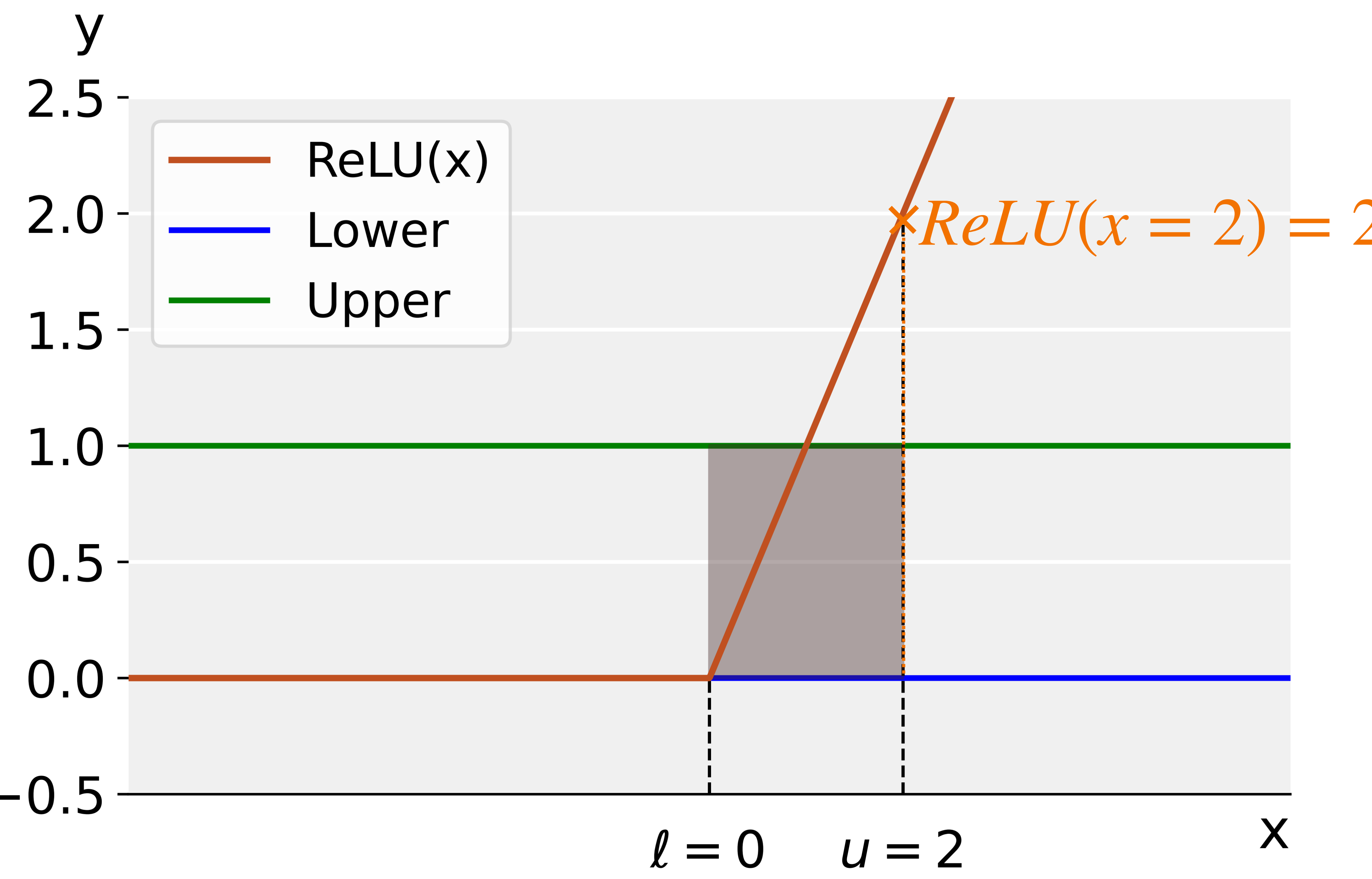
- $ReLU^{\sharp}([l, u]) = [0,1]$



- $[l, u] = [0,\ 0.5]$ ✅
- $[l, u] = [0,1]$ ✅

Learning from Examples

- $ReLU^{\sharp}([l, u]) = [0,1]$

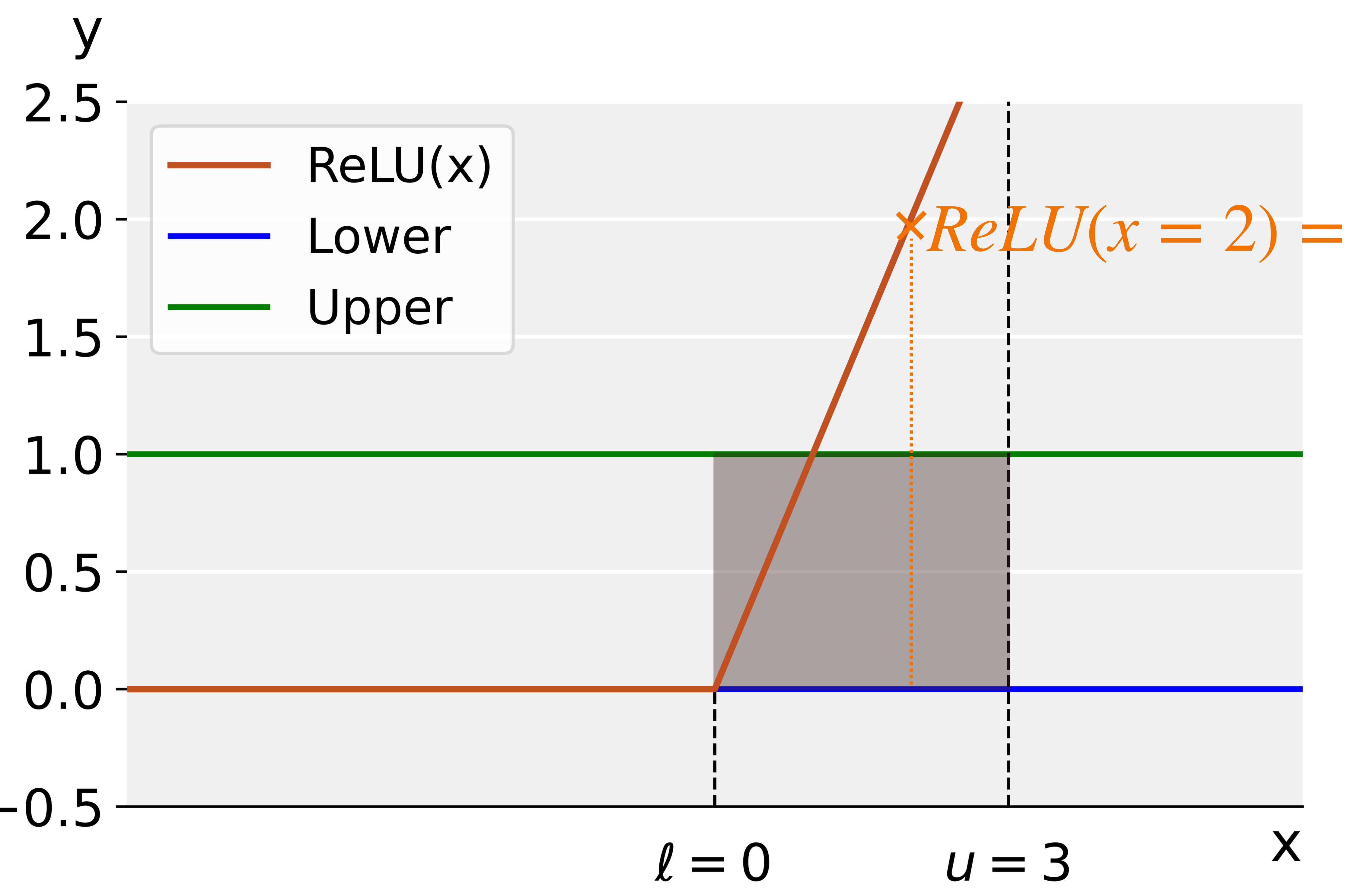


- $[l, u] = [0,\ 0.5]$ ✅
- $[l, u] = [0,1]$ ✅
- $[l, u] = [0,2]$ ❌

When $x = 2 \in \gamma([0,2])$, $ReLU(x) = 2 \notin [0,1]$ …

Learning from Examples

- $ReLU^{\sharp}([l, u]) = [0,1]$

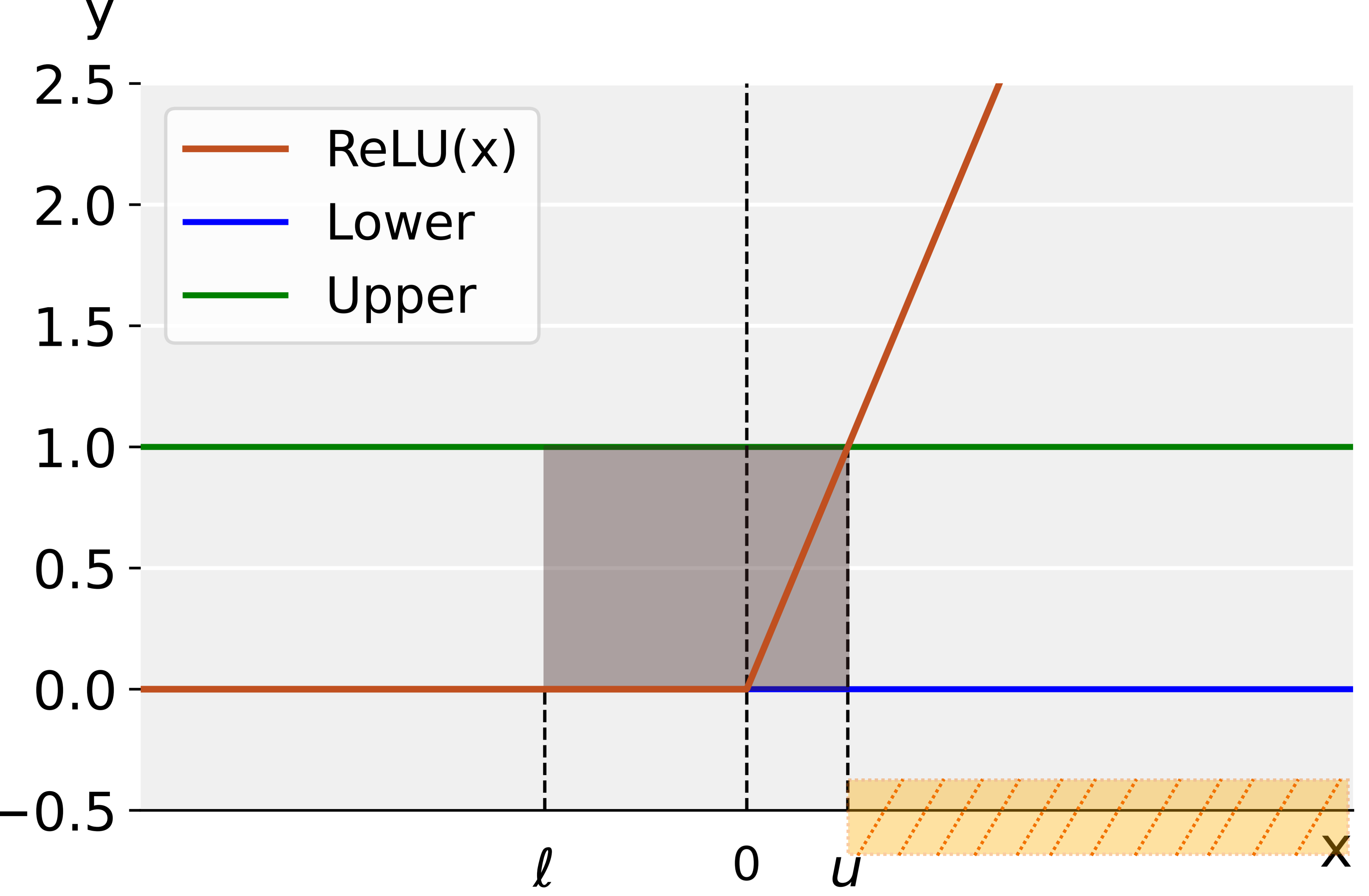


- $[l, u] = [0,\ 0.5]$ ✅
- $[l, u] = [0,1]$ ✅
- $[l, u] = [0,2]$ ❌
- $[l, u] = [0,3]$ ❌

When $x = 2,3 \in \gamma([0,3])$, $ReLU(x) = 2,3 \notin [0,1]$ …

Learning from Examples

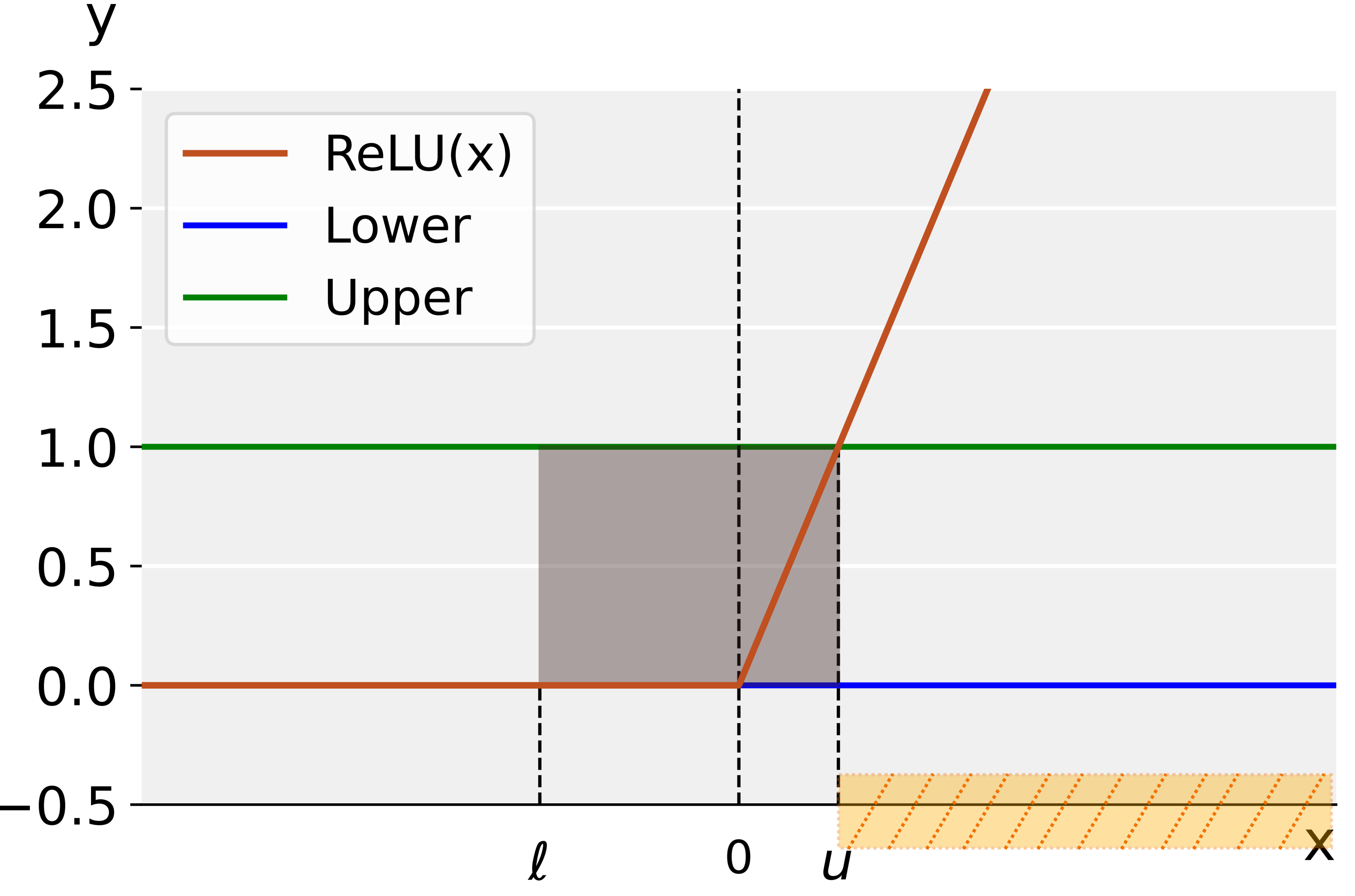
- $ReLU^{\sharp}([l, u]) = [0,1]$



- $[l, u] = [0,\ 0.5]$ ✅
- $[l, u] = [0,1]$ ✅
- $[l, u] = [0,2]$ ❌
- $[l, u] = [0,3]$ ❌
- $[l, u] = [0,4]$ ❌
- …

Learning from Examples

- $ReLU^\sharp([l,u]) = [0,1]$



- $[l,u] = [0,\ 0.5]$ ✅
- $[l,u] = [0,1]$ ✅
- $[l,u] = [0,2]$ ❌
- $[l,u] = [0,3]$ ❌
- $[l,u] = [0,4]$ ❌
- …

Counterexamples

Violating abstract element z, s.t., $z \in A, \quad F(\gamma(z)) \not\subseteq_C \gamma(F^\sharp(z))$

First Step: Aggregate Over Counterexamples

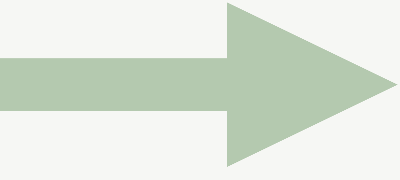
- $[l, u] = [0,2] = c_1$
- $[l, u] = [0,3] = c_2$
- $[l, u] = [0,4] = c_3$
- ...

Counterexample Set $A^* = \{c_1, c_2, c_3, \ldots\}$

First Step: Aggregate Over Counterexamples

- $[l, u] = [0,2]$ = c_1
- $[l, u] = [0,3]$ = c_2
- $[l, u] = [0,4]$ = c_3
- ...

Counterexample Set $A^* = \{c_1, c_2, c_3, \ldots\}$

Measure Unsoundness  **Measure unsoundness degree** (A^*)

First Step: Aggregate Over Counterexamples

- $[l, u] = [0,2] = c_1$
- $[l, u] = [0,3] = c_2$
- $[l, u] = [0,4] = c_3$
- ...

Counterexample Set $A^* = \{c_1, c_2, c_3, \dots\}$

unsoundness degree (A^*) = aggregate (contribution (c_i))

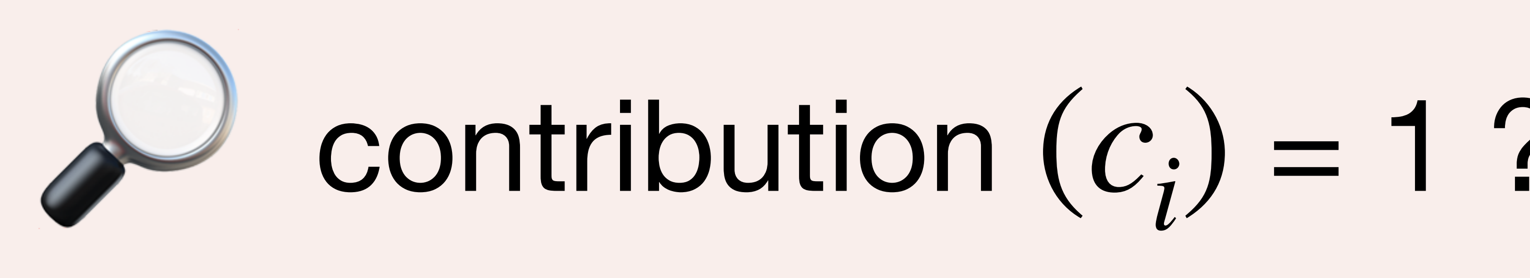
How to Measure Counterexample?

How to Measure Counterexample?

How to Measure Counterexample?

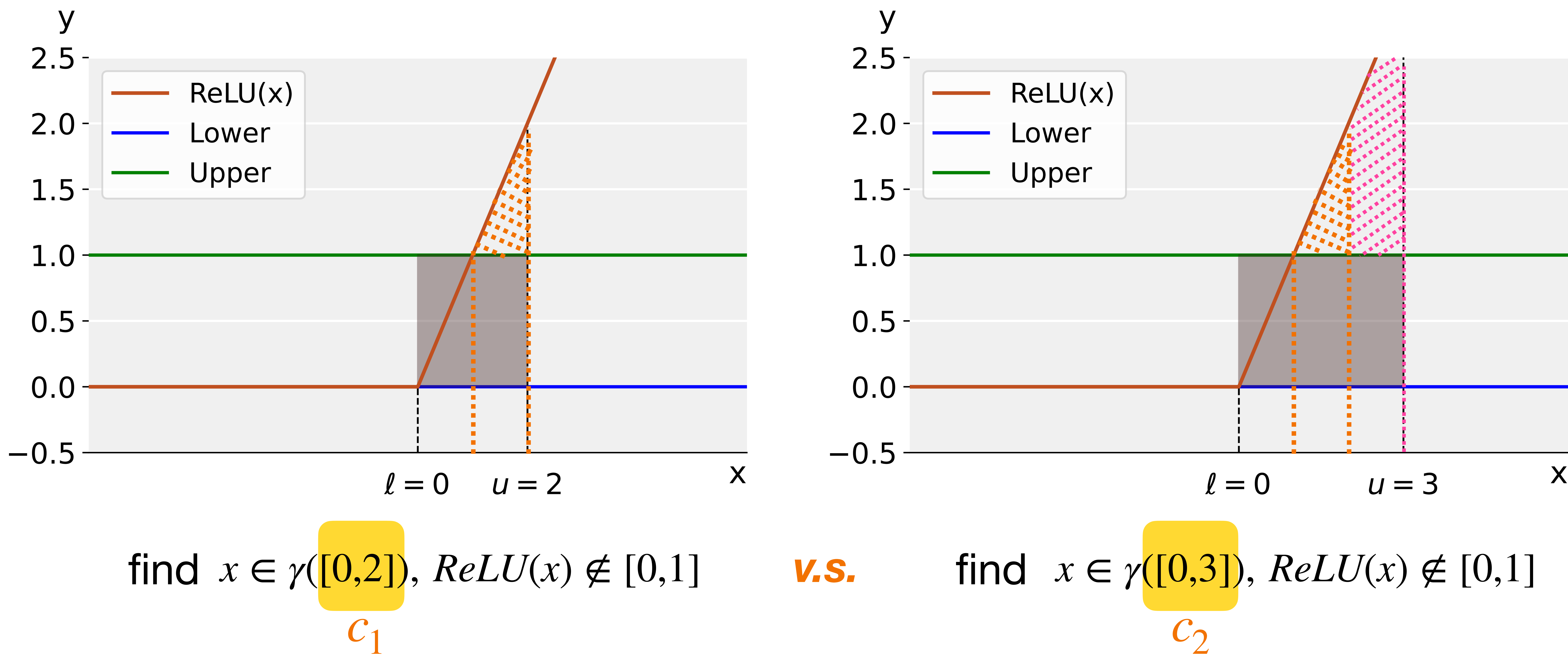
- $A*$ can be infinite.

How to Measure Counterexample?

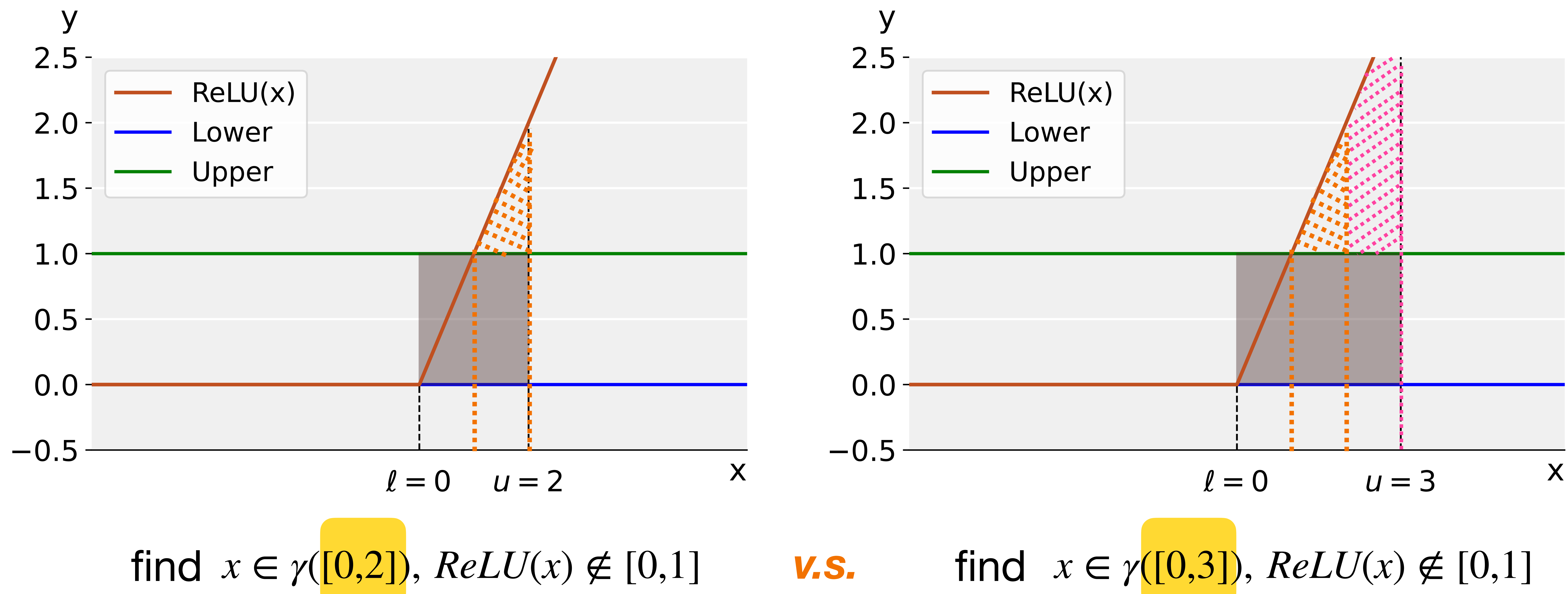


- Fails to capture the **different contributions** of different counterexamples to unsoundness.

Different Contributions of Different Counterexamples

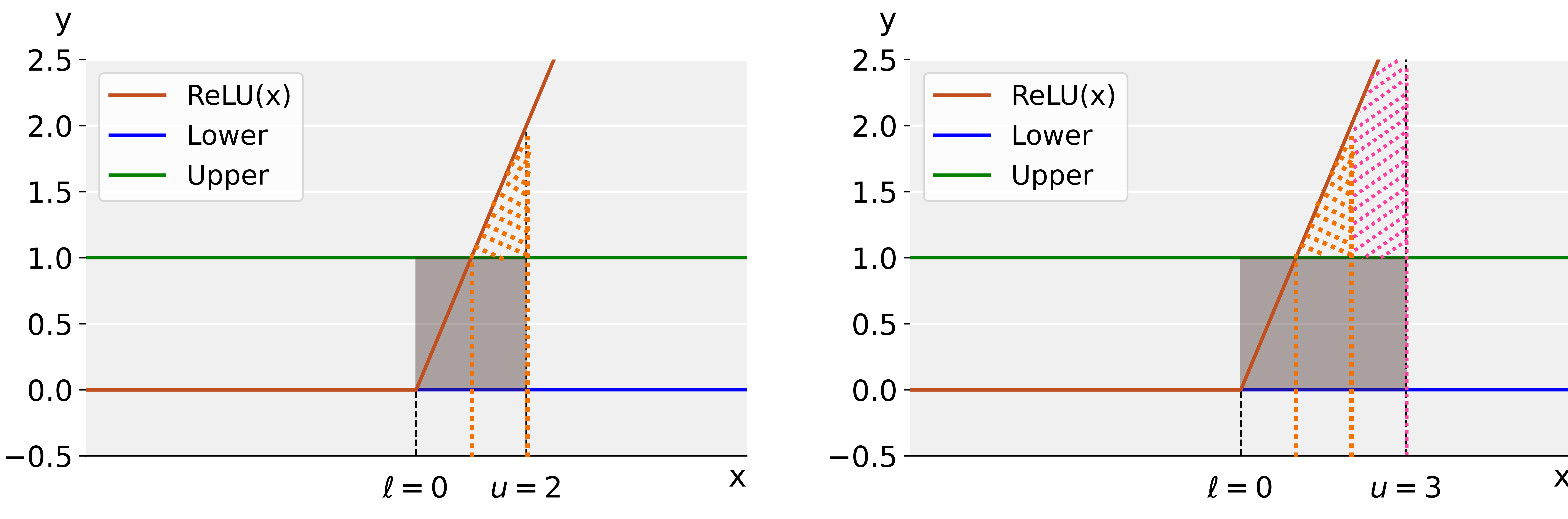


Different Contributions of Different Counterexamples



Each violating abstract element contributes different degrees of unsoundness.

Different Contributions of Different Counterexamples

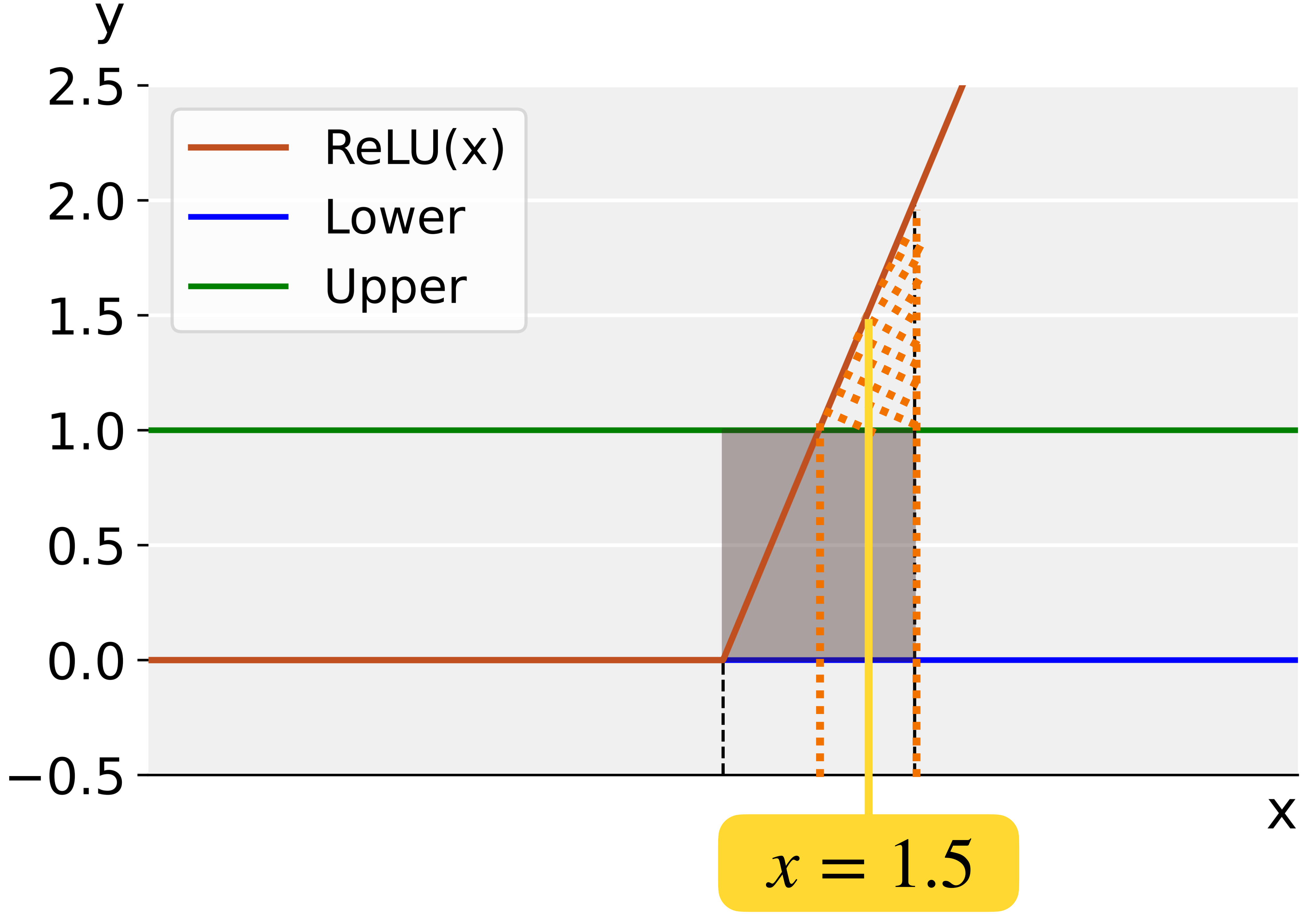


find $x \in \gamma([0,2]),\ ReLU(x) \notin [0,1]$ **v.s.** find $x \in \gamma([0,3]),\ ReLU(x) \notin [0,1]$

contribution (c_i) = aggregate (contribution (violating concrete points))

Third Step: Shape-aware Deviation

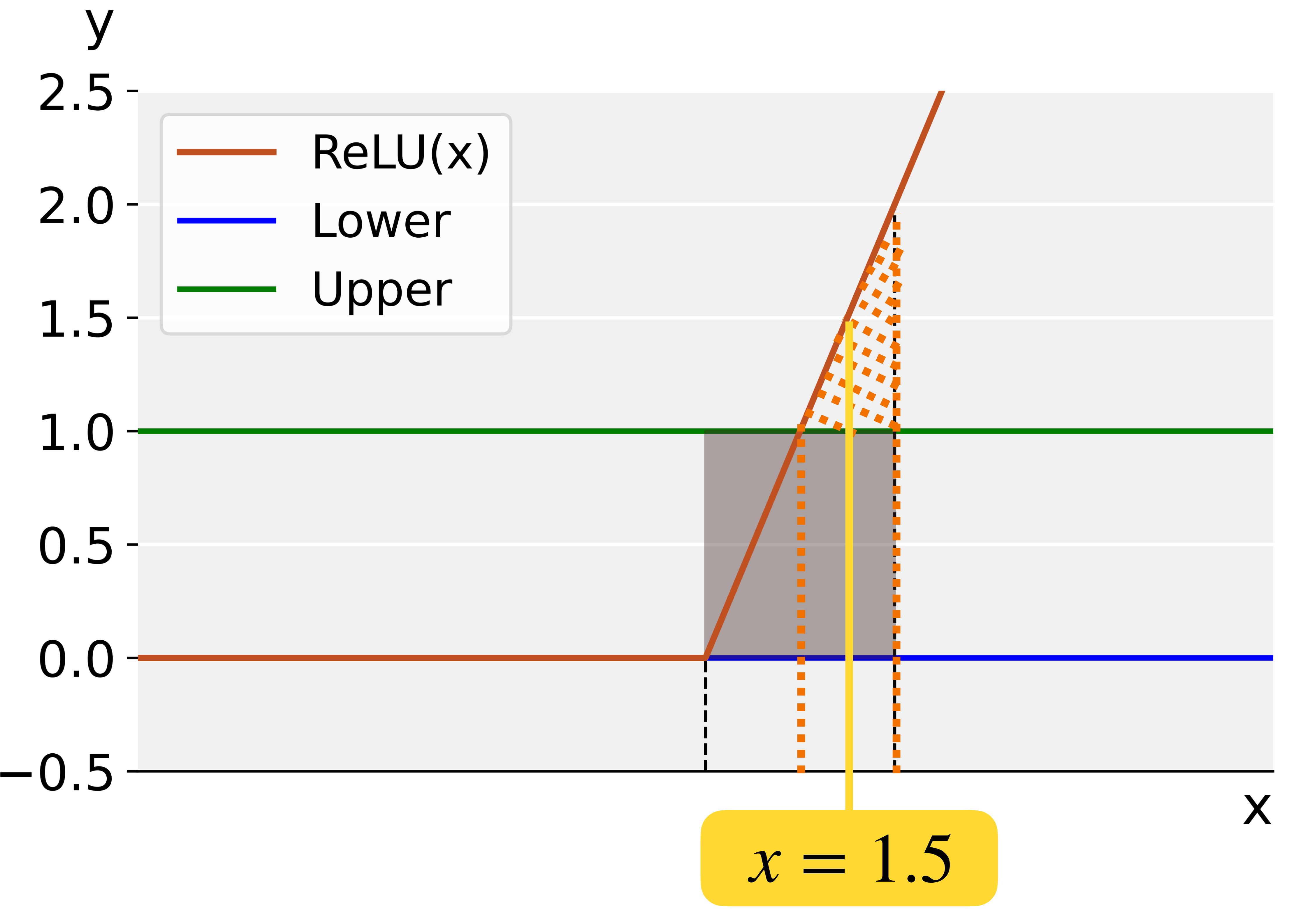
- $ReLU^\sharp([l, u]) = [0,1]$



- counterexample $c_1 = [0,2]$
- violated concrete point $x = 1.5$

Third Step: Shape-aware Deviation

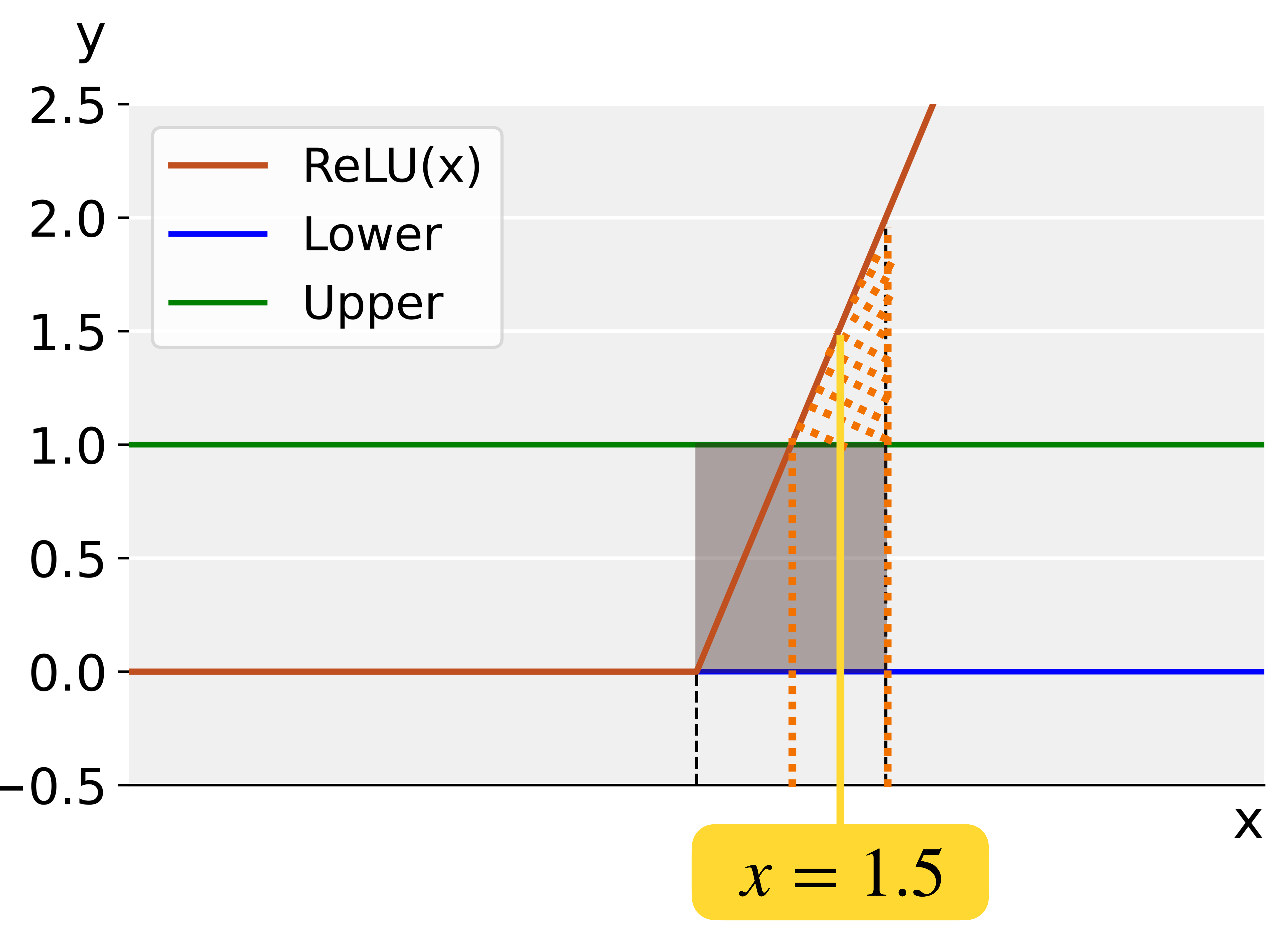
- $ReLU^{\sharp}([l, u]) = [0,1]$



- counterexample $c_1 = [0,2]$
- violated concrete point $x = 1.5$
- $ReLU(x) = 1.5 \notin [0,1]$

Third Step: Shape-aware Deviation

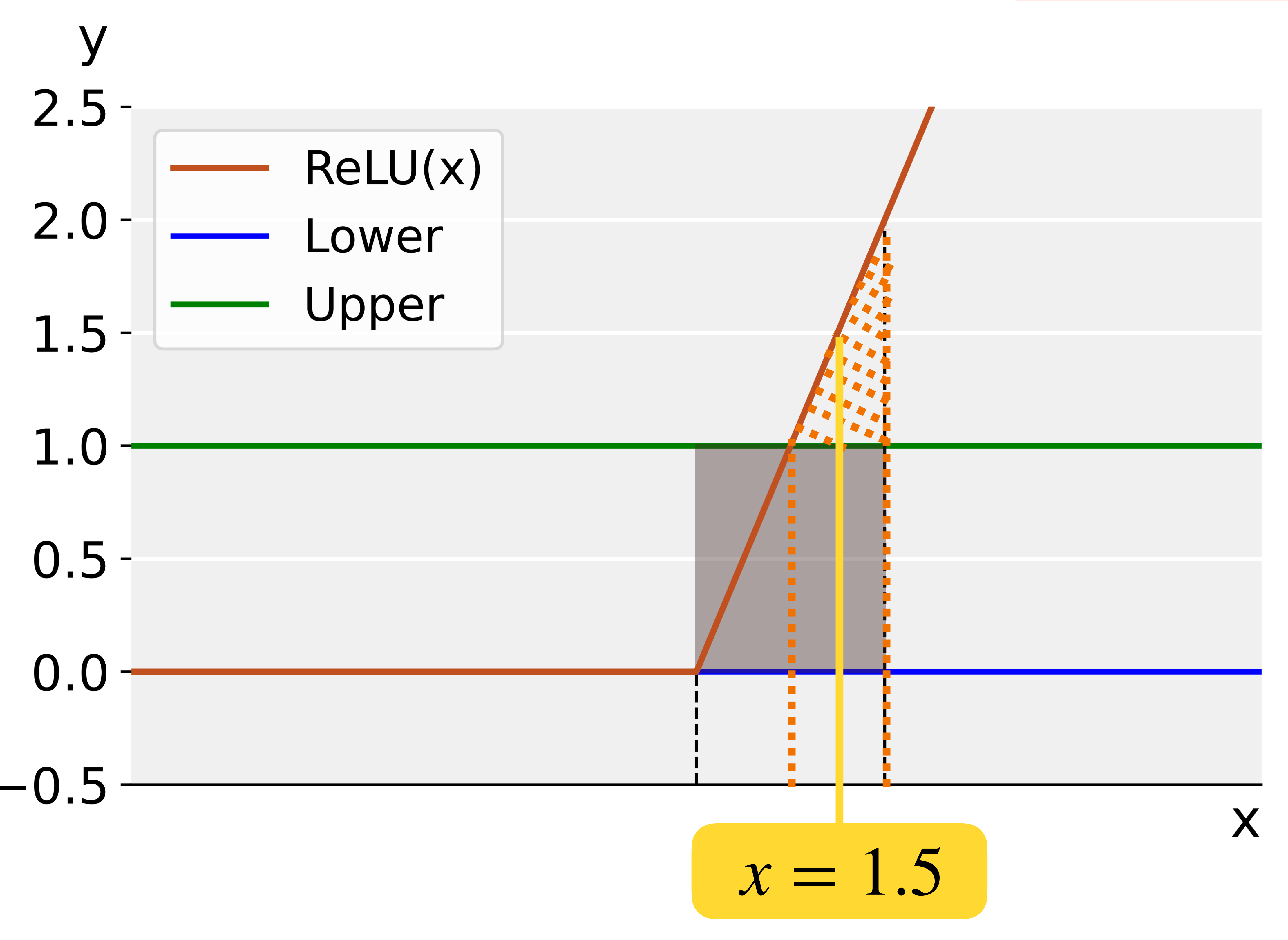
- $ReLU^\sharp([l,u]) = [0,1]$ → **Two constraints:** $ReLU(x) \geq 0 \wedge ReLU(x) \leq 1$



- counterexample $c_1 = [0,2]$
- violated concrete point $x = 1.5$
- $ReLU(x) = 1.5 \notin [0,1]$

Third Step: Shape-aware Deviation

- $ReLU^{\sharp}([l, u]) = [0,1]$ → **Two constraints:** $ReLU(x) \geq 0 \wedge ReLU(x) \leq 1$



- counterexample $c_1 = [0,2]$
- violated concrete point $x = 1.5$
- $ReLU(x) = 1.5 \notin [0,1]$

Third Step: Shape-aware Deviation

- $ReLU^{\sharp}([l, u]) = [0,1]$ → **Two constraints:** $ReLU(x) \geq 0 \wedge ReLU(x) \leq 1$

- **Outputed abstract element from the abstract transformer corresponds to several constraints**
- **Contribution (x) = Measure how much the output of concrete point violate the constraints**

Shape-aware Deviation

Constraint	
ScalarBound	*AffineBound*

Shape-aware Deviation

Constraint	
ScalarBound	***AffineBound***
$\varepsilon(m, y \leq c) := \max(0, m - c)$	$\varepsilon(m, y \leq a_0 + \sum a_i x_i) := \max(0, m - (a_0 + \sum a_i x_i))$
$\varepsilon(m, y \geq c) := \max(0, c - m)$	$\varepsilon(m, y \geq a_0 + \sum a_i x_i) := \max(0, (a_0 + \sum a_i x_i) - m)$

Define the measurement **by distance**.

Shape-aware Deviation

Constraint	
ScalarBound	***AffineBound***
$\varepsilon(m, y \leq c) := \max(0, m - c)$	$\varepsilon(m, y \leq a_0 + \sum a_i x_i) := \max(0, m - (a_0 + \sum a_i x_i))$
$\varepsilon(m, y \geq c) := \max(0, c - m)$	$\varepsilon(m, y \geq a_0 + \sum a_i x_i) := \max(0, (a_0 + \sum a_i x_i) - m)$

- violation (1.5) = $max(0 - 1.5,\ 0) + max(1.5 - 1,\ 0)$

Constraint (0), Output (1.5); Output (1.5), Constraint (1)

Violated Concrete Point (1.5)

Hierarchical Cost Function

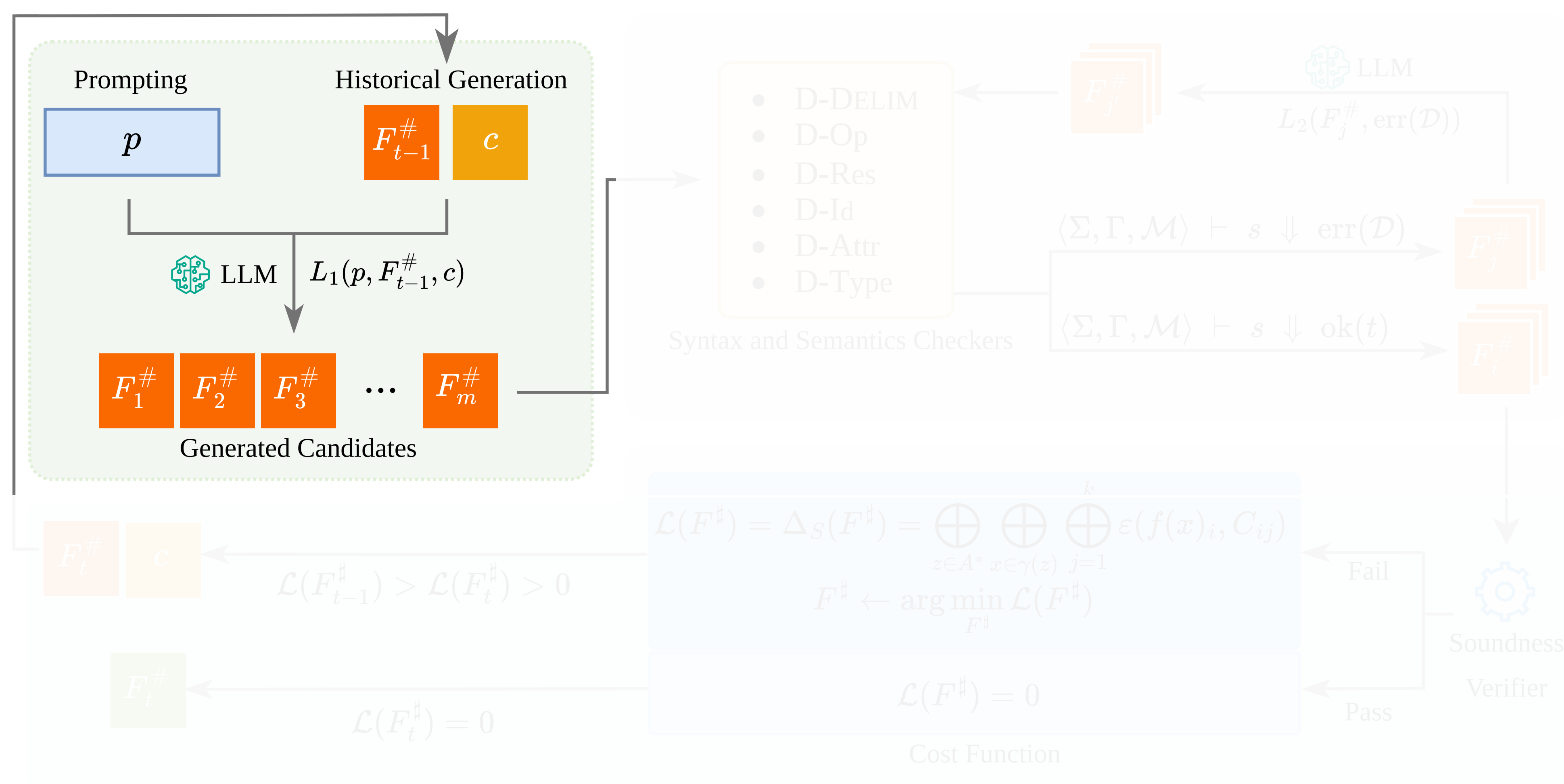
$$\Delta_S(F^\sharp) = {}^{z}\bigoplus_{z \in A^*} {}^{x}\bigoplus_{x \in \gamma(z)} {}^{j}\bigoplus_{j=1}^{k} \varepsilon\big(f(x), C_j\big)$$

→ Over Abstract Elements
→ Over Concrete States
→ Over Constraints

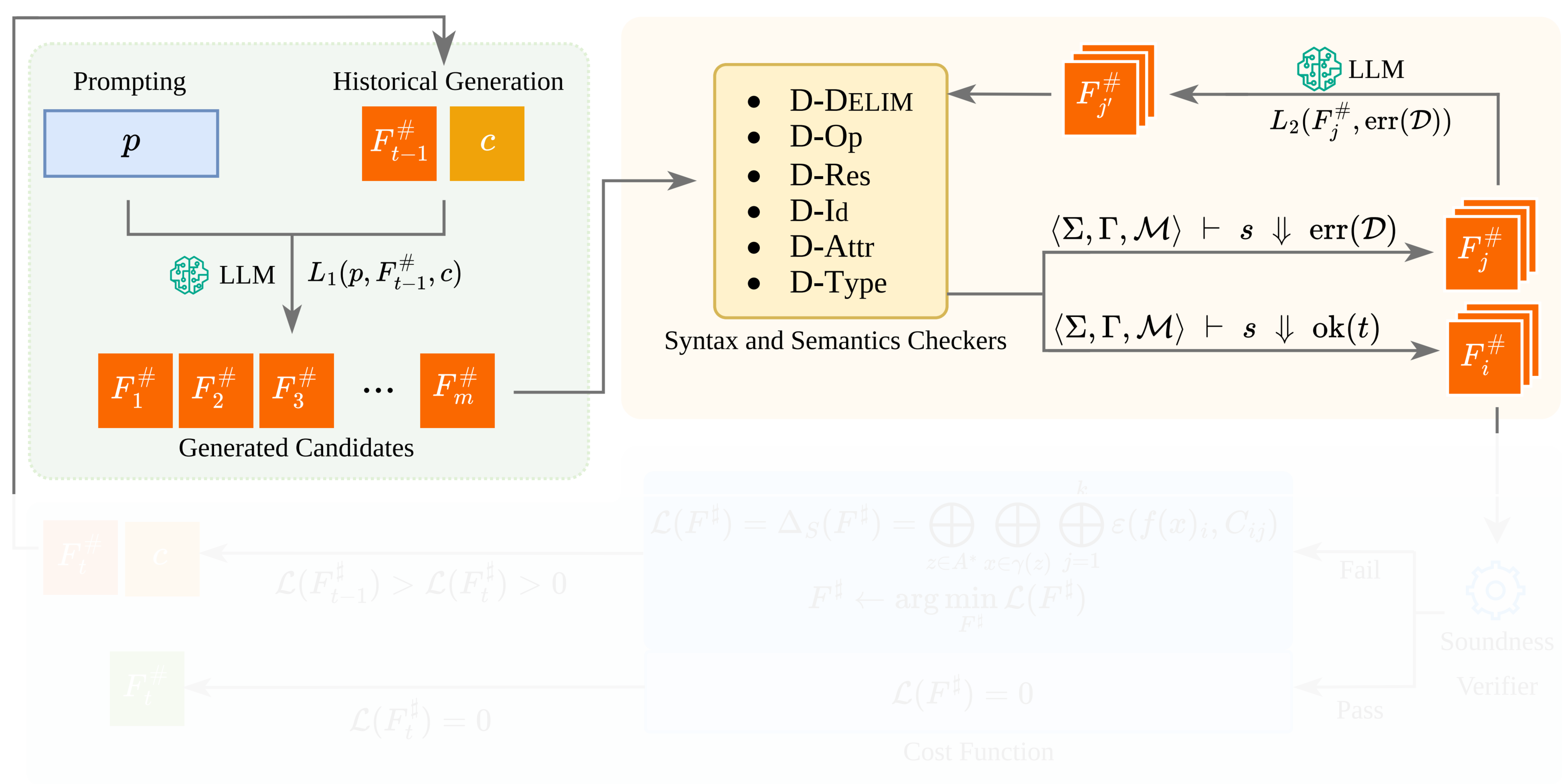
A principled, quantitative, continuous measure.

**In practice we do weighted sampling for infinite set A^* and $\gamma(z)$.*

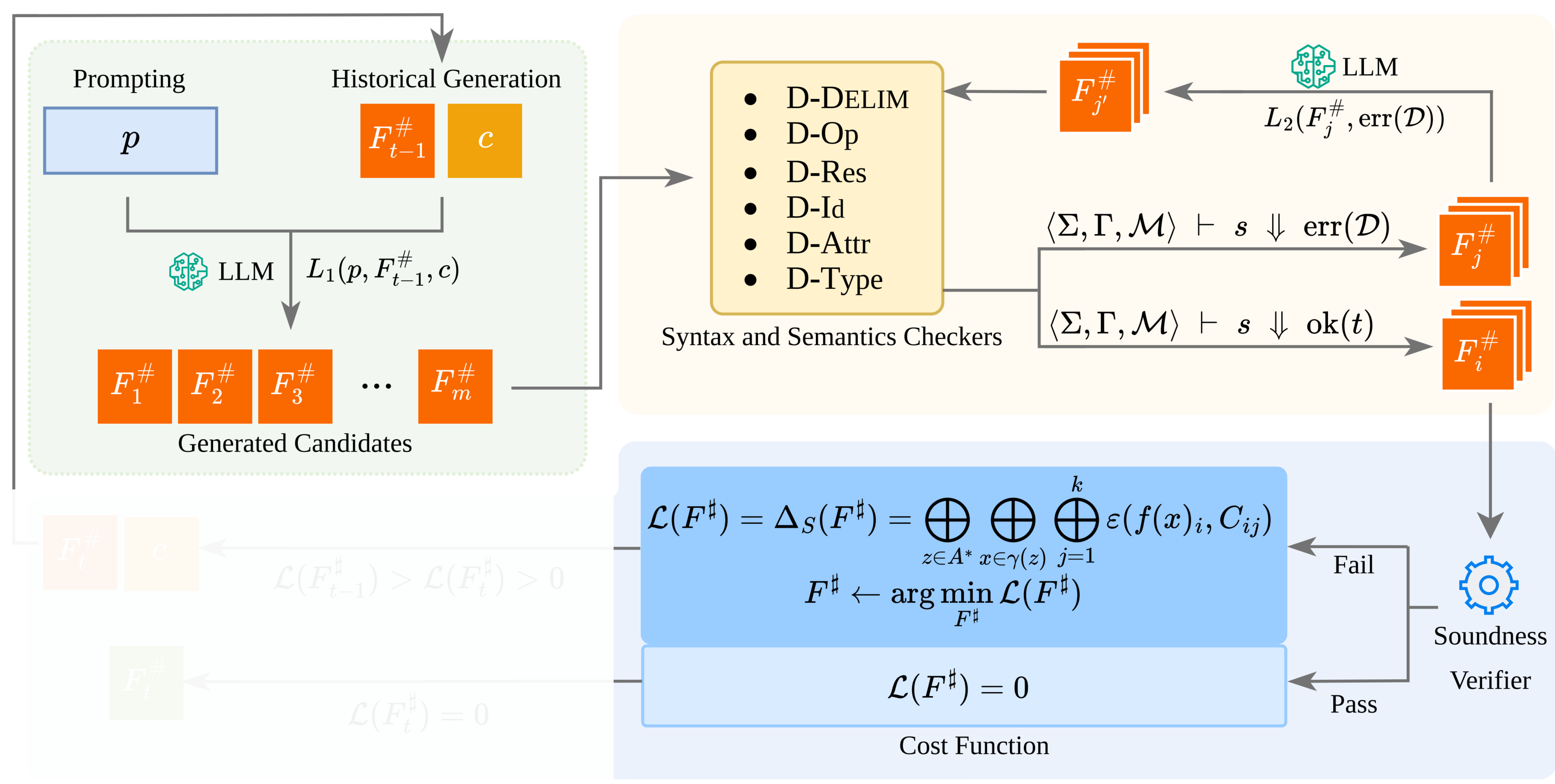
Pipeline of SAIL



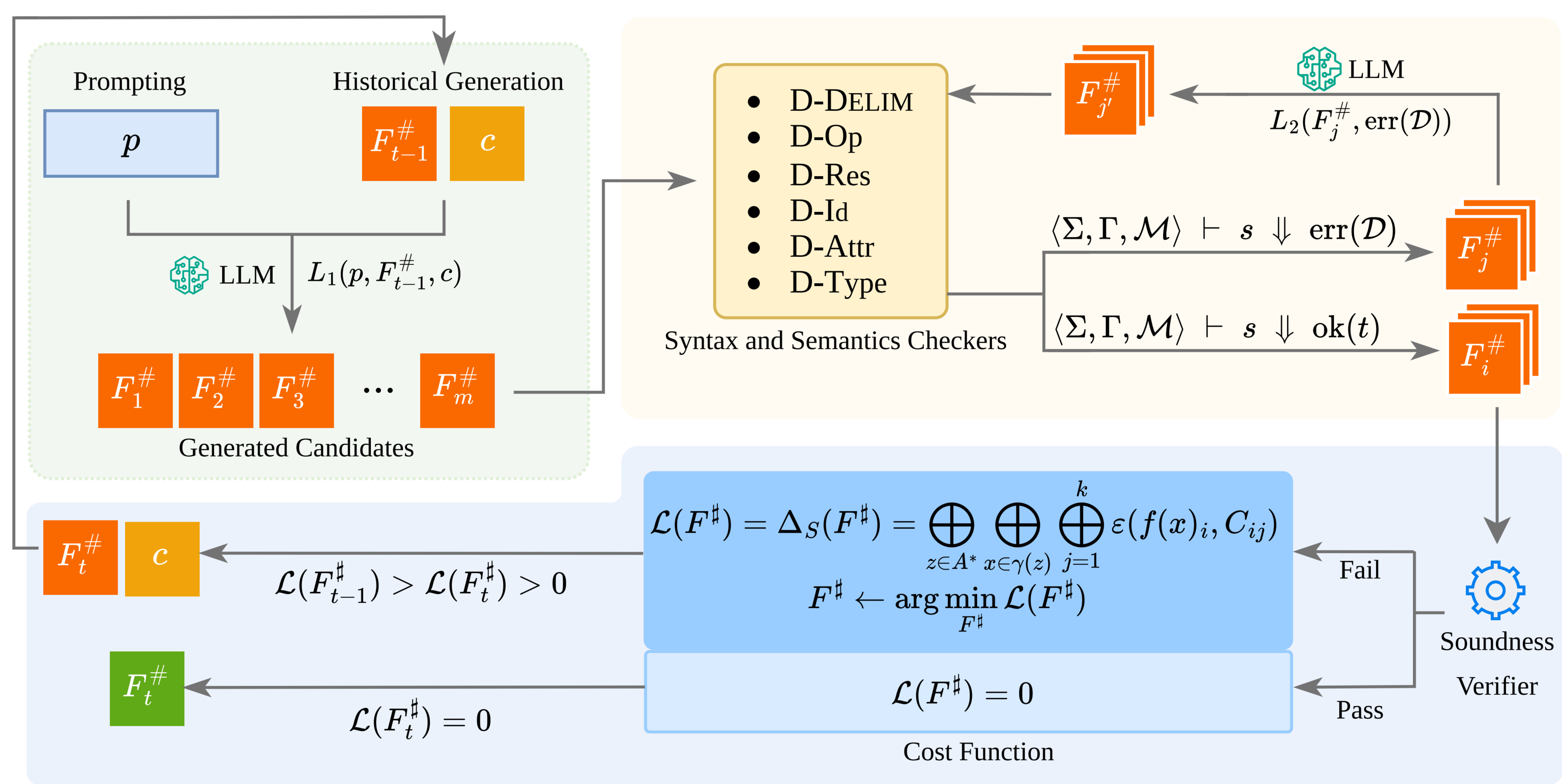
Pipeline of SAIL



Pipeline of SAIL



Pipeline of SAIL



Implicit Effects on Precision

Implicit Effects on Precision

1. Prompting Design

- Few-shot examples with case distinctions
- DSL constructions for common patterns (e.g., tangent line, secant line)

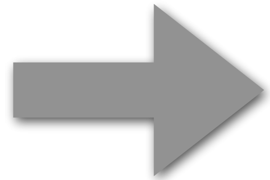
```
[Information about the certifier. Using DeepPoly as an example below:]
DeepPoly certifier uses four kinds of bounds to approximate the operator: (Float l, Float u, PolyExp L, PolyExp U). They must follow the constraints that: curr[l] <= curr <= curr[u] & curr[L] <= curr <= curr[U]. ‘curr‘ here means the current neuron, ‘prev‘ means the inputs to the operator. When the operator takes multiple inputs, use ‘prev_0‘, ‘prev_1‘, ... to refer to each input. So every transformer in each case of the case analysis must return four values. Use any funstions below if needed instead of use arithmetic operators. Function you can use:
- func simplify_lower(Neuron n, Float coeff) = (coeff >= 0) ? (coeff * n[l]) : (coeff * n[u]);
- func simplify_upper(Neuron n, Float coeff) = (coeff >= 0) ? (coeff * n[u]) : (coeff * n[l]);
- func replace_lower(Neuron n, Float coeff) = (coeff >= 0) ? (coeff * n[L]) : (coeff * n[U]);
- func replace_upper(Neuron n, Float coeff) = (coeff >= 0) ? (co
- func priority(Neuron n) = n[layer];
- func priority2(Neuron n, Float c) = -n[layer];
- func stop(Neuron n) = false;
- func stop_traverse(Neuron n, Float c) = false;
- func backsubs_lower(PolyExp e, Neuron n) = (e.traverse(ba
place_lower)e <= n).map(simplify_lower);
- func backsubs_upper(PolyExp e, Neuron n) = (e.traverse(ba
place_upper)e >= n).map(simplify_upper);
- func f(Neuron n1, Neuron n2) = n1[l] >= n2[u];
- func slope(Float x1, Float x2) = ((x1 * (x1 + 3))-(x2 * (x2 + 3)))
- func intercept(Float x1, Float x2) = x1 * ((x1 + 3) / 6) - (slope(x
- func f(Neuron n1, Neuron n2) = n1[l] >= n2[u];
- func f1(Float x) = x < 3 ? x * ((x + 3) / 6) : x;
- func f2(Float x) = x * ((x + 3) / 6);
- func f3(Neuron n) = max(f2(n[l]), f2(n[u]));
- func compute_l(Neuron n1, Neuron n2) = min([n1[l]*n2[l], n
- func compute_u(Neuron n1, Neuron n2) = max([n1[l]*n2[l], n
- func avg(List<Float> xs) = sum(xs) / len(xs);
- func argmax(List<Neuron> ns, (Neuron, Neuron -> Bool) cmp) = [ n | n in ns, forall m in ns. cmp(n, m) ];
- func argmin(List<Neuron> ns, (Neuron, Neuron -> Bool) cmp) = [ n | n in ns, forall m in ns. cmp(n, m) ];

Don't add comments to DSL.

[Two-shot prompting. Using DeepPoly as an example below:]
### Example: Abs operator
Input: Generate the transformer for ‘abs‘ operator
Output:
"""
def Shape as (Float l, Float u, PolyExp L, PolyExp U)
{[(curr[l]<=curr),(curr[u]>=curr),(curr[L]<=curr),(curr[U]>=curr)]};
```

```
transformer deeppoly Abs -> ((prev[l]) >= 0) ? ((prev[l]), (prev[u]), (prev), (prev)) : (((prev[u]) <= 0) ? (0-(prev[u]), 0-(prev[l]), 0-(prev), 0-(prev)) : (0, max(prev[u], 0-prev[l]), prev, prev*(prev[u]+prev[l])/(prev[u]-prev[l]) - (((2*prev[u])*prev[l])/(prev[u]-prev[l]))) );
"""

### Example: Affine operator
Input: Generate the transformer for ‘affine‘ operator
Output:
"""
def Shape as (Float l, Float u, PolyExp L, PolyExp U)
{[(curr[l]<=curr),(curr[u]>=curr),(curr[L]<=curr),(curr[U]>=curr)]};
transformer deeppoly Affine -> (backsubs_lower(prev.dot(curr[weight]) + curr[bias], curr), backsubs_upper(prev.dot(curr[weight]) + curr[bias], curr), prev.dot(curr[weight]) + curr[bias], prev.dot(curr[weight]) + curr[bias]);
"""

[Query.]
### Now generate the transformer for ‘{api}‘ operator
Input: Generate the transformer for ‘{api}‘ operator
Output: """
```

Implicit Effects on Precision

2. Implicit enforcement by cost function

- Weight function penalize violations near transition points more

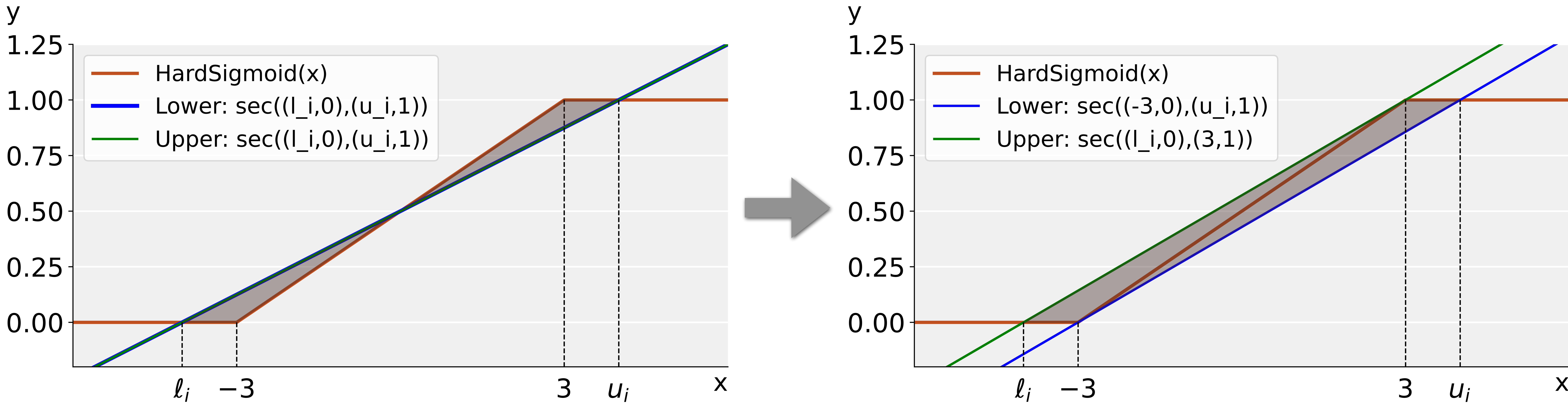


Correct case distinctions

Implicit Effects on Precision

2. Implicit enforcement by cost function

- Cost function penalize affine bounds with misaligned slopes or intercepts



Evaluation

Efficiency

Generality

Scalability

Evaluation

Efficiency

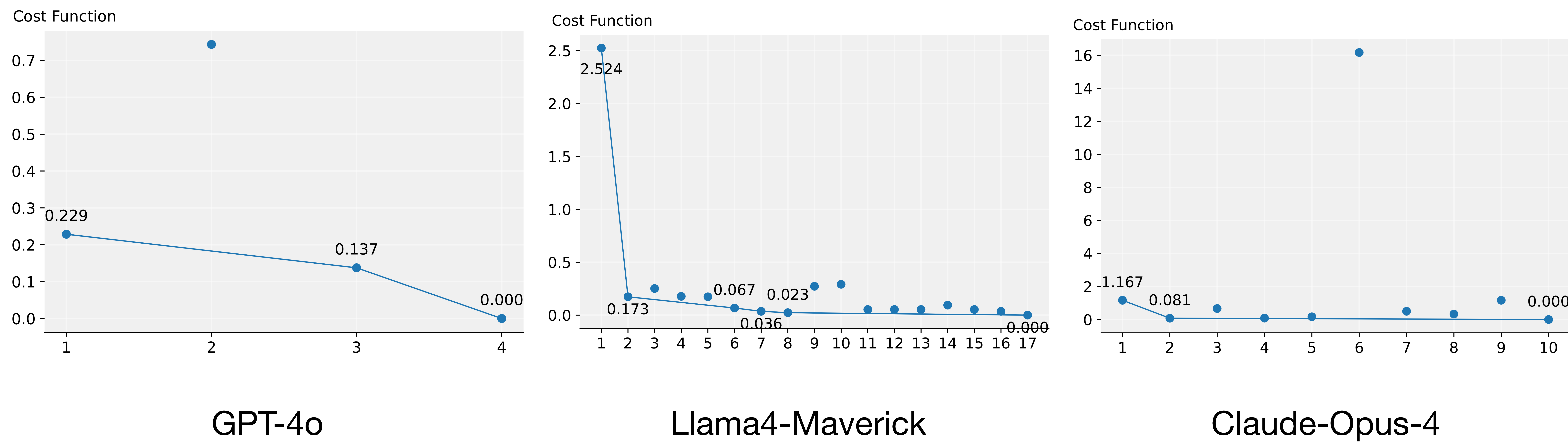
- Each synthesis finished within ~2h
- Cost function dramatically improves synthesis success
- Consistently achieves high precision

Generality

Scalability

Guidance of Cost Function

- Different LLMs synthesize HardSigmoid abstract transformer in DeepPoly.



Synthesized HardSigmoid Transformer

```
1  transformer DeepPoly {
2      HardSigmoid ->
3          (prev[u] <= -3) ?
4              (0, 0, 0, 0) :
5          (prev[l] >= 3) ?
6              (1, 1, 1, 1) :
7          (prev[l] >= -3) ?
8              (prev[u] <= 3) ?
9                  ((prev[l] / 6) + 0.5,
10                  (prev[u] / 6) + 0.5,
11                  (prev / 6) + 0.5,
12                  (prev / 6) + 0.5) :
13                 ((prev[l] / 6) + 0.5,
14                  1,
15                  (((0.5 - (prev[l] / 6)) / (prev[u] - prev[l])) * prev)
16                    + ((prev[l] / 6) + 0.5)
17                    - (((0.5 - (prev[l] / 6)) / (prev[u] - prev[l])) * prev[l]),
18                  1) :
19         (prev[u] <= 3) ?
20             (0,
21              (prev[u] / 6) + 0.5,
22              0,
23              (((((prev[u] / 6) + 0.5) / (prev[u] - prev[l])) * prev)
24                - ((((prev[u] / 6) + 0.5) / (prev[u] - prev[l])) * prev[l]))) :
25             (0, 1, 0, 1);
26 }
```

PyTorch Code of HardSigmoid

```python
def HardSigmoid(self, abs_elem, prev, curr, poly_size, curr_size, prev_size, input_size, batch_size):
    cse_var_95 = SparseTensor([], [], 0, torch.tensor([]), dense_const=6, type= type(6)).unsqueeze(0).unsqueeze(1).repeat(
    cse_var_104 = abs_elem.get_elem('l', prev)
    cse_var_105 = SparseTensor([], [], 0, torch.tensor([]), dense_const=3, type= type(3)).unsqueeze(0).unsqueeze(1).repeat
    cse_var_91 = binary(binary(cse_var_104, cse_var_105, operator.add), cse_var_95, operator.truediv)
    cse_var_103 = SparseTensor([], [], 0, torch.tensor([]), dense_const=-3, type= type(-3)).unsqueeze(0).unsqueeze(1).repe
    cse_var_96 = binary(cse_var_104, cse_var_103, operator.ge)
    cse_var_106 = abs_elem.get_elem('u', prev)
    cse_var_97 = binary(cse_var_106, cse_var_105, operator.le)
    cse_var_324 = SparseTensor([], [], 0, torch.tensor([]), dense_const=1.0, type= type(1.0)).unsqueeze(0).unsqueeze(1).re
    cse_var_99 = binary(cse_var_104, cse_var_105, operator.ge)
    cse_var_100 = binary(cse_var_106, cse_var_103, operator.le)
    cse_var_102 = SparseTensor([], [], 0, torch.tensor([]), dense_const=0, type= type(0)).unsqueeze(0).unsqueeze(1).repeat
    cse_var_101 = where(cse_var_96, cse_var_91, cse_var_102)
    l_new = where(cse_var_100, cse_var_102, where(cse_var_99, cse_var_324, where(cse_var_97, cse_var_101, cse_var_101)))
    cse_var_94 = binary(binary(cse_var_106, cse_var_105, operator.add), cse_var_95, operator.truediv)
    rewrite_new_14 = convert_to_float(cse_var_96)
    u_new = where(cse_var_100, cse_var_102, where(cse_var_99, cse_var_324, where(cse_var_97, where(cse_var_96, cse_var_94,
    cse_var_93 = prev.convert_to_poly(abs_elem)
    cse_var_92 = SparseTensor([], [], 0, torch.tensor([]), dense_const=3, type= type(3)).unsqueeze(0).unsqueeze(1).repeat(
    cse_var_172 = cse_var_93.get_const()
    cse_var_173 = cse_var_93.get_mat(abs_elem)
    cse_var_254 = binary(cse_var_172, cse_var_92, operator.add)
    cse_var_75 = binary(cse_var_106, cse_var_104, operator.sub)
    cse_var_90 = binary(binary(cse_var_324, cse_var_91, operator.sub), cse_var_75, operator.truediv)
    cse_var_170 = repeat(cse_var_172, torch.tensor([batch_size, 1]))
    cse_var_171 = repeat(cse_var_173, torch.tensor([batch_size, 1, 1]))
    cse_var_256 = binary(cse_var_90, cse_var_170, operator.mul)
    cse_var_257 = binary(repeat(cse_var_90.unsqueeze(2), torch.tensor([1, 1, poly_size])), cse_var_171, operator.mul)
    cse_var_79 = PolyExpSparse(abs_elem.network, 0.0, cse_var_324)
    cse_var_63 = cse_var_79.get_const()
    cse_var_70 = repeat(cse_var_97.unsqueeze(2), torch.tensor([1, 1, poly_size]))
    cse_var_71 = cse_var_79.get_mat(abs_elem)
```

```python
    cse_var_72 = repeat(cse_var_99.unsqueeze(2), torch.tensor([1, 1, poly_size]))
    cse_var_73 = repeat(cse_var_100.unsqueeze(2), torch.tensor([1, 1, poly_size]))
    cse_var_82 = PolyExpSparse(abs_elem.network, 0.0, cse_var_102)
    cse_var_77 = cse_var_82.get_const()
    cse_var_255 = binary(cse_var_256, binary(cse_var_91, binary(cse_var_90, cse_var_104, operator.mul), operator.sub), ope
    cse_var_84 = cse_var_82.get_mat(abs_elem)
    cse_var_85 = repeat(cse_var_96.unsqueeze(2), torch.tensor([1, 1, poly_size]))
    cse_var_245 = where(cse_var_96, cse_var_255, cse_var_77)
    cse_var_247 = where(cse_var_85, cse_var_257, cse_var_84)
    cse_var_88 = SparseTensor([], [], 0, torch.tensor([]), dense_const=6, type= type(6)).unsqueeze(0).unsqueeze(1).repeat(
    cse_var_89 = SparseTensor([], [], 0, torch.tensor([]), dense_const=6, type= type(6)).unsqueeze(0).unsqueeze(1).unsquee
    cse_var_250 = binary(cse_var_254, cse_var_88, operator.truediv)
    cse_var_252 = binary(cse_var_173, cse_var_89, operator.truediv)
    cse_var_249 = repeat(cse_var_250, torch.tensor([batch_size, 1]))
    cse_var_246 = where(cse_var_96, cse_var_249, cse_var_249)
    cse_var_251 = repeat(cse_var_252, torch.tensor([batch_size, 1, 1]))
    cse_var_248 = where(cse_var_85, cse_var_251, cse_var_251)
    cse_var_243 = where(cse_var_97, cse_var_246, cse_var_245)
    cse_var_244 = where(cse_var_70, cse_var_248, cse_var_247)
    cse_var_241 = where(cse_var_99, cse_var_63, cse_var_243)
    cse_var_242 = where(cse_var_72, cse_var_71, cse_var_244)
    L_new = PolyExpSparse(abs_elem.network, where(cse_var_73, cse_var_84, cse_var_242) , where(cse_var_100, cse_var_77, cs
    cse_var_74 = binary(cse_var_94, cse_var_75, operator.truediv)
    cse_var_239 = binary(cse_var_74, cse_var_170, operator.mul)
    cse_var_240 = binary(repeat(cse_var_74.unsqueeze(2), torch.tensor([1, 1, poly_size])), cse_var_171, operator.mul)
    cse_var_66 = PolyExpSparse(abs_elem.network, 0.0, SparseTensor([], [], 0, torch.tensor([]), dense_const=1, type= type(
    cse_var_234 = where(cse_var_96, cse_var_249, repeat(cse_var_66.get_const(), torch.tensor([batch_size, 1])))
    cse_var_236 = where(cse_var_85, cse_var_251, repeat(cse_var_66.get_mat(abs_elem), torch.tensor([batch_size, 1, 1])))
    cse_var_238 = binary(cse_var_239, binary(cse_var_74, cse_var_104, operator.mul), operator.sub)
    cse_var_235 = where(cse_var_96, cse_var_249, cse_var_238)
    cse_var_237 = where(cse_var_85,
    cse_var_232 = where(cse_var_97,   (variable) cse_var_237: SparseTensor
    cse_var_233 = where(cse_var_70, cse_var_237, cse_var_236)
    cse_var_230 = where(cse_var_99, cse_var_63, cse_var_232)
    cse_var_231 = where(cse_var_72, cse_var_71, cse_var_233)
    U_new = PolyExpSparse(abs_elem.network, where(cse_var_73, cse_var_84, cse_var_231) , where(cse_var_100, cse_var_77, cs
    return l_new, u_new, L_new, U_new
```

Evaluation: Precision

Dataset	Network	Training	Activation	Layers	Perturbation ϵ	Precision	
						Our work	Handcrafted
MNIST	Conv	Standard	ReLU	6	0.005	1.0000	1.0000
	Conv	Standard	ReLU6	3	0.005	1.0000	✗
	FCN_5×100	DiffAI	HardTanh	5	0.005	0.9500	0.9500
	FCN_6×500	PGD	HardSigmoid	6	0.005	1.0000	✗
	FCN_3×100	Standard	GELU	3	0.005	0.9400	✗
	FCN_4×1024	Standard	GELU	4	0.005	1.0000	✗
	FCN_3×100	Standard	ELU	3	0.005	0.1400	✗
CIFAR10	Conv	DiffAI	ReLU	3	8/255	1.0000	1.0000
	FCN_4×100	Standard	ReLU6	4	8/255	0.4490	✗
	FCN_7×1024	Standard	HardTanh	7	8/255	0.9231	0.9231
	FCN_4×100	Standard	HardSwish	4	8/255	0.1154	✗
	FCN_6×500	PGD	HardSigmoid	6	8/255	1.0000	✗
	FCN_6×500	PGD	GELU	6	8/255	1.0000	✗
	FCN_7×1024	Standard	GELU	7	8/255	0.9787	✗

Evaluation

Efficiency

- Each synthesis finished within ~2h
- Cost function dramatically improves synthesis success
- Consistently achieves high precision

Generality

- Diverse abstract transformers for diverse domains

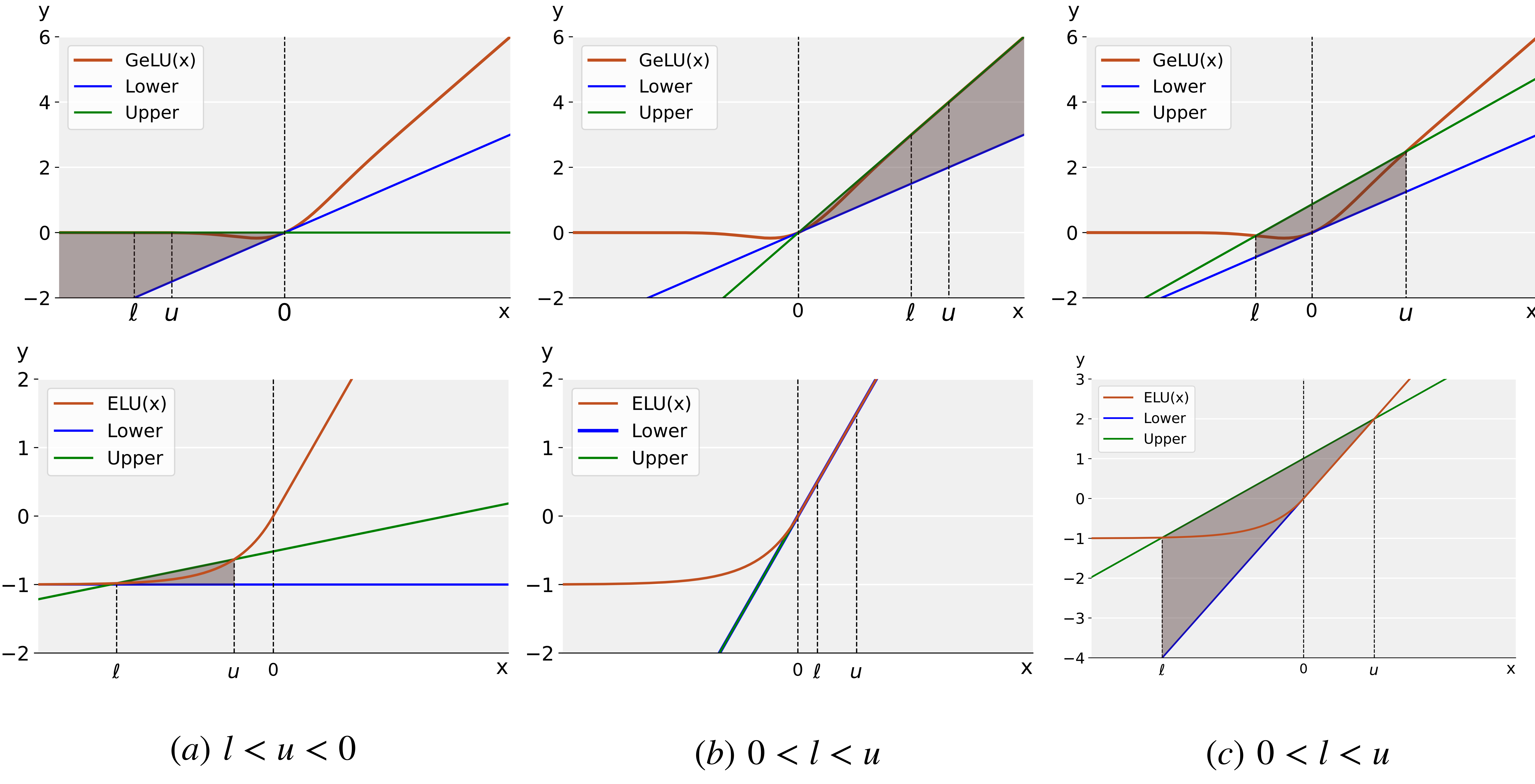
GeLU, Sigmoid, HardSigmoid, HardTanh …

DeepPoly, DeepZ, Box …

- Transformers for first-order derivatives

Scalability

GeLU and ELU - DeepPoly



(a) $l < u < 0$ (b) $0 < l < u$ (c) $0 < l < u$

Evaluation

Efficiency

- Each synthesis finished within ~2h
- Cost function dramatically improves synthesis success
- Consistently achieves high precision

Generality

- Diverse abstract transformers for diverse domains

GeLU, Sigmoid, HardSigmoid, HardTanh …

DeepPoly, DeepZ, Box …

- Transformers for first-order derivatives

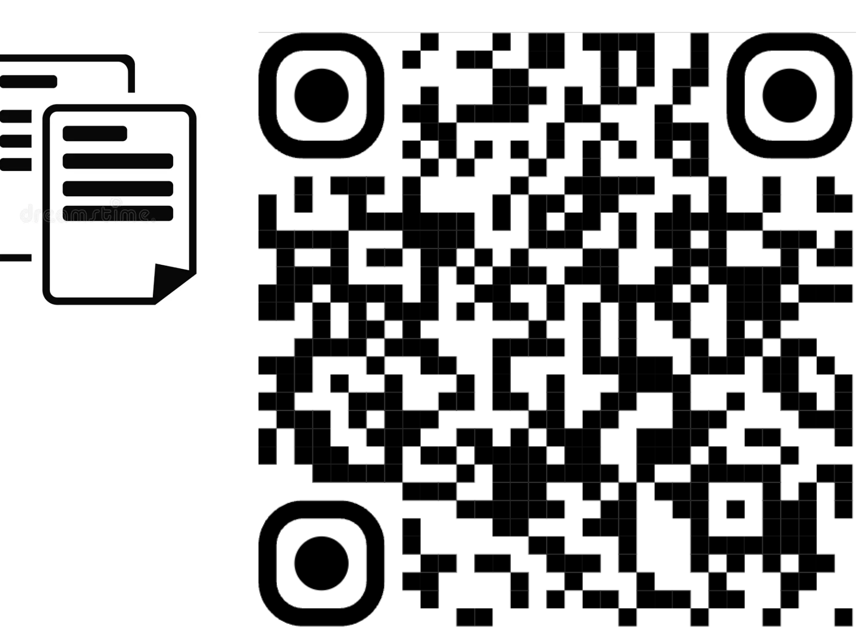
Scalability

- 650 MB peak memory usage
- Synthesize once, reuse forever

Thanks!

We hope our work can inspire efficient abstract interpretation and make it more accessible to everyone!

More details:



Acknowledgments: