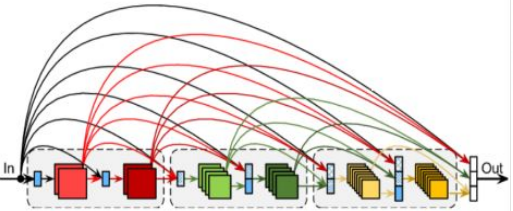
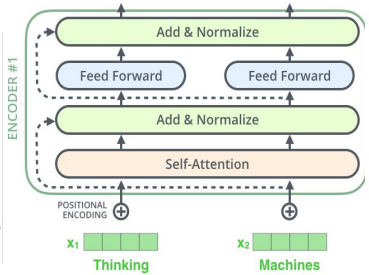
AAAI 2022 Tutorial on Neural Network Verification

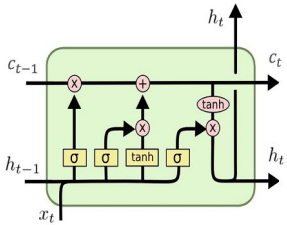
Part III: Practical Incomplete Verification with `auto_LiRPA`

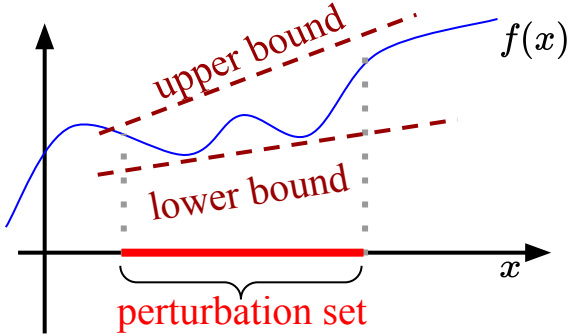
Huan Zhang (CMU), **Kaidi Xu (Drexel)**, Shiqi Wang (Columbia) and Cho-Jui Hsieh (UCLA)

Feb 23, 2022









Goal of This Tutorial

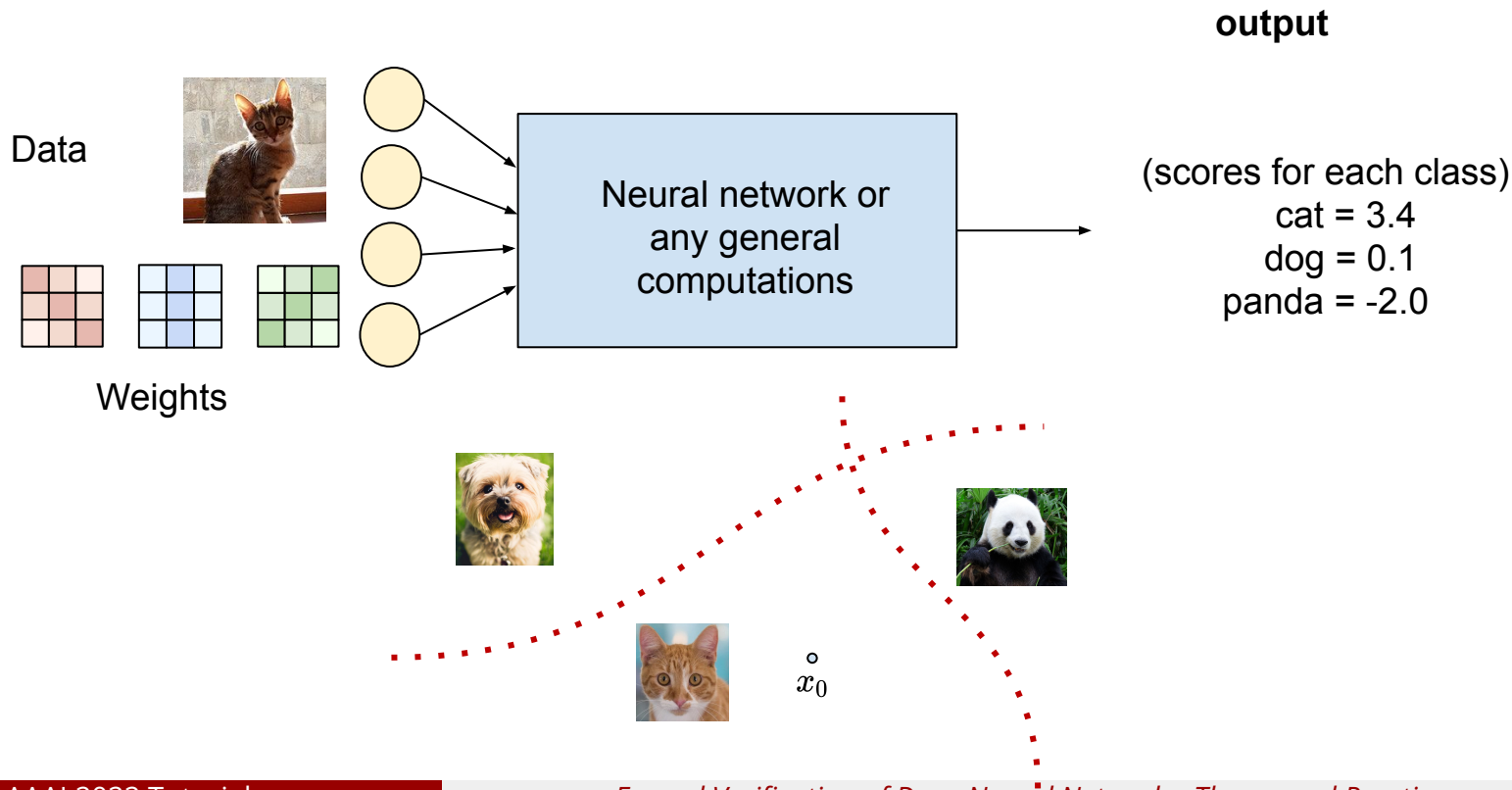
How do we verify an existing DNN model practically?

Tools to introduce in this tutorial:

- Incomplete verifier toolbox: **auto_LiRPA**
- Complete verifier toolbox: ***α,β*-CROWN**

⭐Please **star** our repos if you find them useful.😊

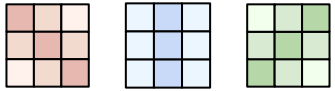
auto_LiRPA: A Robustness Verification Library



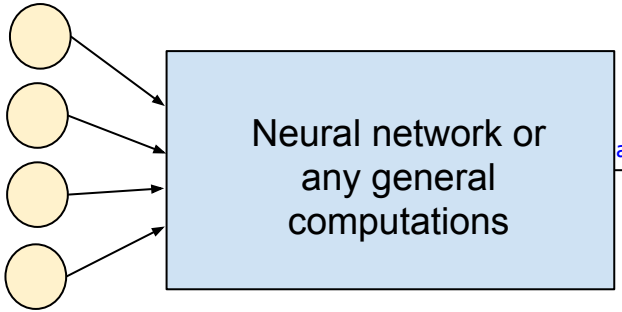
auto_LiRPA: A Robustness Verification Library

data or model weights under perturbations

Data perturbed arbitrarily within a set



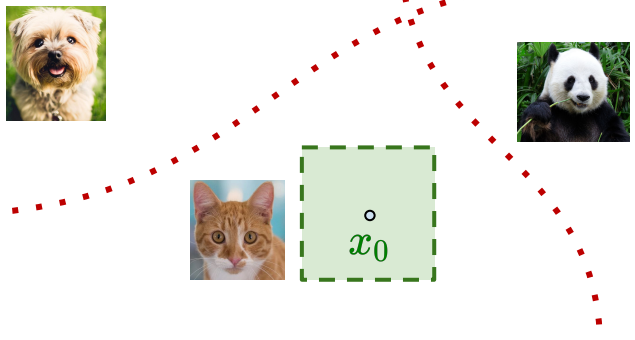
or allowing weights to slightly deviate from the original



output bounds

(guaranteed score ranges)
2.3 ≤ cat ≤ 4.5
-0.8 ≤ dog ≤ **1.2**
-4.2 ≤ panda ≤ **-0.1**

we guarantee that "cat" stays top-1 under input/weight perturbations



Benefits of `auto_LiRPA`

- Automatic bound derivation on existing PyTorch models - users need no knowledge of verification algorithms. No manual derivation or implementation for new model architectures.

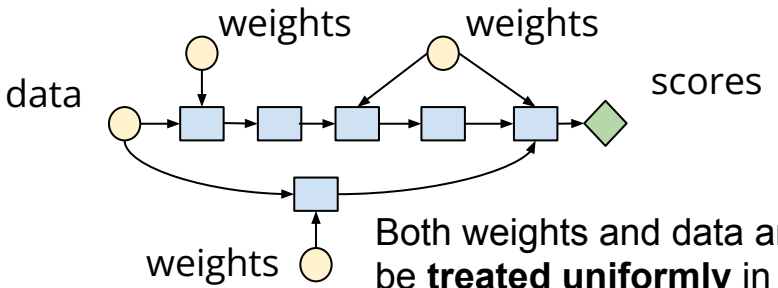
  A **superset** of many previous works on verification (including certified defenses) on different architectures:

  - CNNs (Wong & Kolter 2017)
  - ResNet (Wong et al., 2018)
  - DenseNet (Mirman et al. 2019)
  - LSTM (Ko et al. 2019)
  - Transformers (Shi et al. 2020)
  - ...

Benefits of `auto_LiRPA`

- Automatic bound derivation on existing PyTorch models - users need no knowledge of verification algorithms. No manual derivation or implementation for new model architectures.

- Allows general perturbation analysis on data input or network weights (as they are treated similarly as input of graph)



Both weights and data are inputs for the computational graph, and can be **treated uniformly** in our framework.

Benefits of `auto_LiRPA`

- Automatic bound derivation on existing PyTorch models - users need no knowledge of verification algorithms. No manual derivation or implementation for new model architectures.

- Allows general perturbation analysis on data input or network weights (as they are treated similarly as input of graph)

- Bounds are differentiable and accelerated on GPUs, allowing efficient training for large-scale certified defense

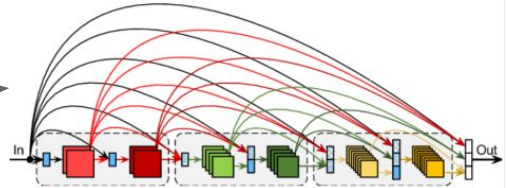
High Level APIs

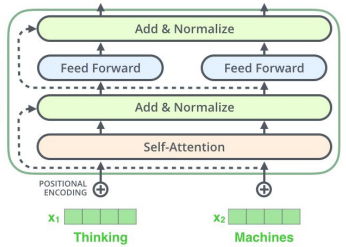
```
from auto_LiRPA import BoundedModule, BoundedTensor, PerturbationLpNorm

# Define computation as a nn.Module.
class MyModel(nn.Module):
    def forward(self, x):
        # Define your computation here.

model = MyModel()
my_input = load_a_batch_of_data()
# Wrap the model with auto_LiRPA.
model = BoundedModule(model, my_input)
# Define perturbation. Here we add Linf perturbation to input data.
ptb = PerturbationLpNorm(norm=np.inf, eps=0.1)
# Make the input a BoundedTensor with the pre-defined perturbation.
my_input = BoundedTensor(my_input, ptb)
# Regular forward propagation using BoundedTensor works as usual.
prediction = model(my_input)
# Compute LiRPA bounds using the backward mode bound propagation (CROWN).
lb, ub = model.compute_bounds(x=(my_input,), method="CROWN")
```

Any complex or irregular network can go here!

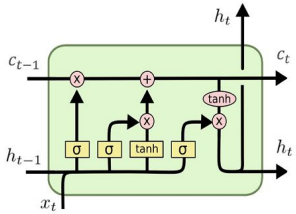
Compute bounds directly!

DenseNet, ResNeXt, WideResNet





Transformers



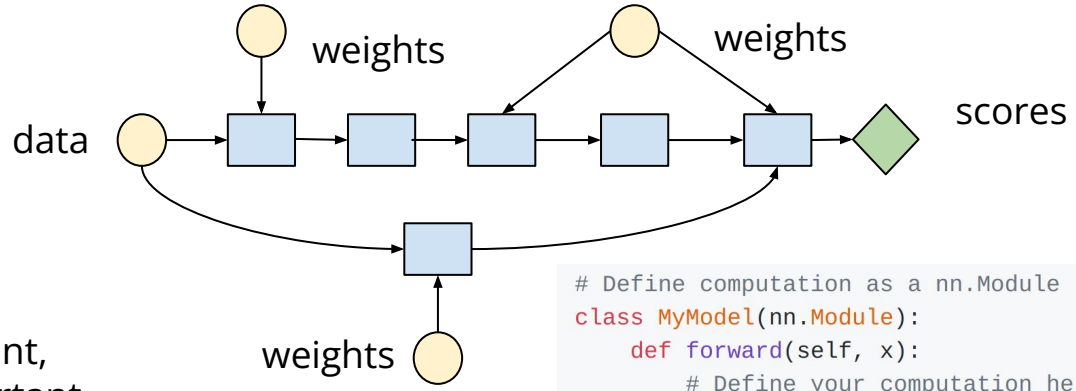
LSTMs

The `BoundedModule` object

- `BoundedModule` is a wrapper of `nn.Module`
- NNs defined by `nn.Module` can be converted to `BoundedModule` (1 line of code), and `BoundedModule` provides LiRPA analysis
- `BoundedModule` builds a *trace graph* given an `nn.Module` and an input tensor. Then it constructs the computational graph to compute bounds based on the trace graph

```
model = MyModel()
my_input = load_a_batch_of_data()
# Wrap the model with auto_LiRPA
model = BoundedModule(model, my_input)
```

shape is important,
contents not important




```
# Define computation as a nn.Module
class MyModel(nn.Module):
    def forward(self, x):
        # Define your computation here
```

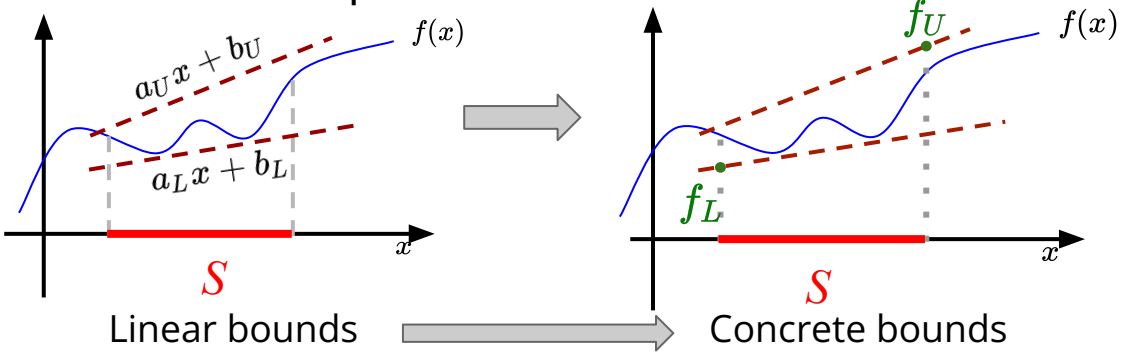
The `PerturbationLpNorm` object

- Defines Lp norm perturbations, p = 0, 1, 2, inf

```
# Define perturbation. For example: Linf norm with perturbation range 0.1.
ptb = PerturbationLpNorm(norm=np.inf, eps=0.1)
```

```
# Define perturbation. For example: Linf norm with customized perturbation range.
ptb = PerturbationLpNorm(norm=np.inf, eps=None, x_L=data_lb, x_U=data_ub)
```

- Other kinds of perturbations can also be user defined as long as a "concretization function" is provided.



The `BoundedTensor` object

- Wraps an input tensor with a perturbation object

```
# Make the input a BoundedTensor with perturbation
my_input = BoundedTensor(my_input, ptb)
```

- Similarly, we have an `BoundedParameter` object for perturbed model parameters. It works similarly to `BoundedTensor`, except that it also registers itself as a model parameter.

The `compute_bounds()` method

- `BoundedModule` object has a `compute_bounds()` method to obtain output bounds given a `BoundedTensor` as input

```
# Compute LiRPA bounds using the backward mode bound propagation (CROWN).
lb, ub = model.compute_bounds(x=(my_input,), method="CROWN")
```

- Supported bounding method:
  - "CROWN"
  - "IBP"
  - "CROWN-IBP"
  - "forward"
  - "alpha-CROWN"

`auto_LiRPA` Demo

Colab Demo:

PaperCode.cc/AutoLiRPA-Tutorial

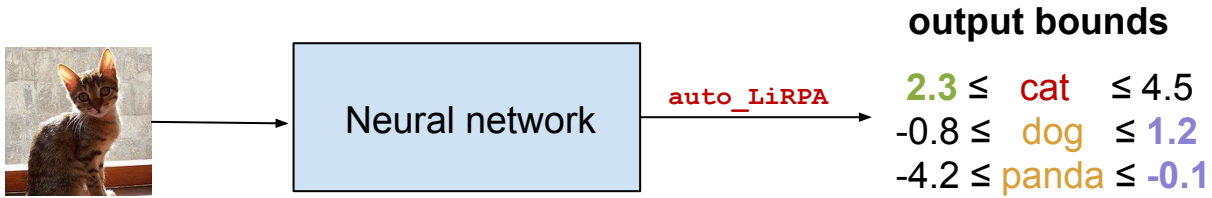
Demo includes an example of computing bounds for a **18-layer ResNet** model on **CIFAR-10** dataset. Once the ResNet model is defined as usual in Pytorch, obtaining provable output bounds is as easy as obtaining gradients through autodiff. Bounds are efficiently computed on GPUs and are differentiable.

`auto_LiRPA` example: Train a Robust Model

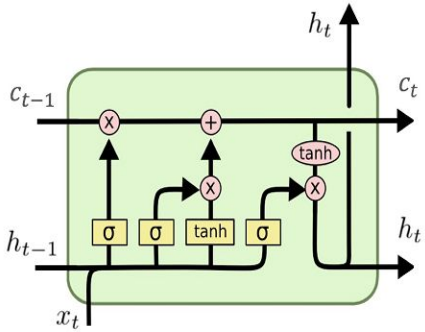
Goal: Train models that they are more verifiably robust

Loss function: enlarge the margin between lower bound of true label and upper bounds of others

- examples/vision/simple_training.py



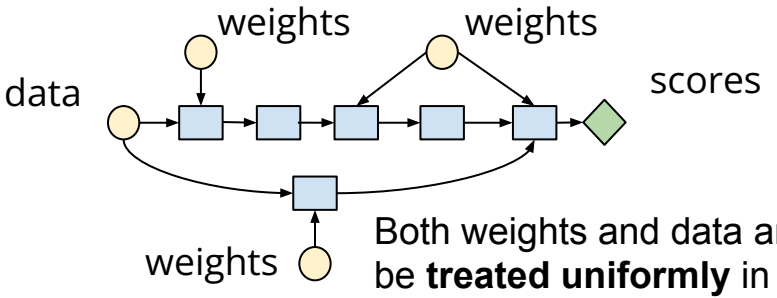
`auto_LiRPA` example: Train a Robust LSTM



In this example we show how easily we can train a verifiably robust LSTM model using `auto_LiRPA`.

- examples/language/lstm.py

`auto_LiRPA` example: Perturbation on Weights



Both weights and data are inputs for the computational graph, and can be **treated uniformly** in our framework.

In this example we show how easily we can train a model with flat optimization landscape by considering perturbations on model weights using `auto_LiRPA`.

- examples/vision/weight_perturbation_training.py

`auto_LiRPA` other examples:

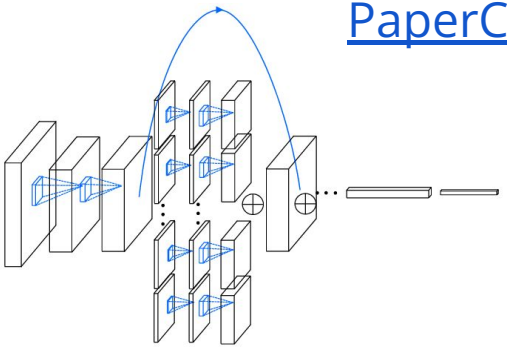
- Many other examples can be found at PaperCode.cc/AutoLiRPA-Examples
  - Basic verification with CROWN/IBP/CROWN-IBP
  - Robust MNIST/CIFAR-10 training
  - Robust ImageNet and TinyImageNet training (multi-GPU)
  - Robust language classifiers using LSTMs
  - Robust language classifiers using Transformers
  - … and send us a pull request for any interesting examples you made!

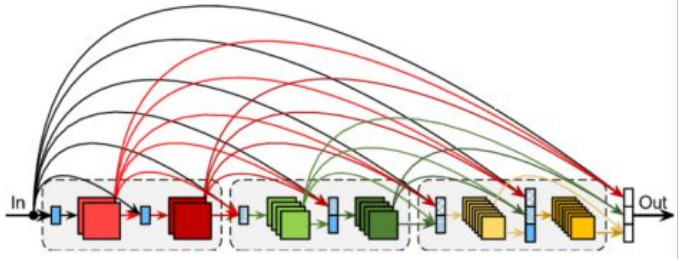
Pretrained Robust Models

- We provide pretrained models than are verifiably robust for CIFAR-10, TinyImageNet and ImageNet (64*64)
- Pretrained models include a large range of modern model architectures, including WideResNet, DenseNet and ResNeXt
- Models achieve state-of-the-art verified accuracy

Models available at
PaperCode.cc/AutoLiRPA-Models

⭐Please **star** our repo if you find it useful.😊





Other papers using `auto_LiRPA`

- Zhang, Huan, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. "Robust deep reinforcement learning against adversarial perturbations on state observations." Advances in Neural Information Processing Systems 33 (2020): 21024-21037.
- Xu, Kaidi, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. "Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers." International Conference on Learning Representations (2021).
- Wang, Shiqi, Kevin Eykholt, Taesung Lee, Jiyong Jang, and Ian Molloy. "Adaptive Verifiable Training Using Pairwise Class Similarity." The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)
- Bharadhwaj, Homanga, De-An Huang, Chaowei Xiao, Anima Anandkumar, and Animesh Garg. "Auditing ai models for verified deployment under semantic specifications." arXiv preprint arXiv:2109.12456 (2021).
- Lorenz, Tobias, Marta Kwiatkowska, and Mario Fritz. "Backdoor Attacks on Network Certification via Data Poisoning." arXiv preprint arXiv:2108.11299 (2021).
- Lyu, Zhaoyang, Minghao Guo, Tong Wu, Guodong Xu, Kehuan Zhang, and Dahua Lin. "Towards evaluating and training verifiably robust neural networks." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4308-4317. 2021.
- Wang, Shiqi, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. "Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification." Advances in Neural Information Processing Systems 34 (2021).
- Zhang, Chong, Jieyu Zhao, Huan Zhang, Kai-Wei Chang, and Cho-Jui Hsieh. "Double Perturbation: On the Robustness of Robustness and Counterfactual Bias Evaluation." In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 3899-3916. 2021.
- Paulsen, Brandon, and Chao Wang. "LinSyn: Synthesizing Tight Linear Bounds for Arbitrary Neural Network Activation Functions." International Conference on Tools and Algorithms for the Construction and Analysis of Systems (2022)
- Sun, Chuangchuang, Dong-Ki Kim, and Jonathan P. How. "ROMAX: Certifiably Robust Deep Multiagent Reinforcement Learning via Convex Relaxation." arXiv preprint arXiv:2109.06795 (2021).
- Chen, Shaoru, Eric Wong, J. Zico Kolter, and Mahyar Fazlyab. "DeepSplit: Scalable Verification of Deep Neural Networks via Operator Splitting." arXiv preprint arXiv:2106.09117 (2021).
- Wang, Yihan, Zhouxing Shi, Quanquan Gu, and Cho-Jui Hsieh. "On the Convergence of Certified Robust Training with Interval Bound Propagation." In International Conference on Learning Representations. 2021.
- …

Next Part

***α*,*β*-CROWN**: Award winning complete verifier with SOTA performance