Automated Algorithm Configuration of α,β-CROWN

Konstantin Kaulen[1] and Holger H. Hoos[1,2]

[1] Chair of AI Methodology, RWTH Aachen University, Germany
[2] LIACS, Leiden University, The Netherlands
{kaulen,hh}@aim.rwth-aachen.de

Abstract. While neural networks have achieved remarkable success across a wide range of application domains, their predictions remain brittle when confronted with adversarial perturbations. To enable their responsible deployment in safety-critical settings, neural network verification techniques have been developed to formally establish input–output properties of interest. However, even state-of-the-art verification systems such as α,β-CROWN, the winner of the Verification of Neural Networks Competition (VNN-COMP) since 2021, may fail to prove challenging properties within reasonable time budgets. Fully exploiting the capabilities of such systems typically demands careful tuning of numerous parameters, a process that often relies on substantial domain expertise and extensive manual experimentation. In this work, for the first time, we apply automated algorithm configuration to α,β-CROWN using `SMAC3`, thereby eliminating the need for labour-intensive manual tuning. We show that, given a carefully designed parameter search space, automatically discovered configurations can achieve performance comparable to expert-crafted configurations, even when using relatively modest computational budgets for the optimisation process. We evaluate our approach on the benchmarks from the regular track of VNN-COMP 2025 and demonstrate that the automatically configured version of α,β-CROWN achieves a higher overall score in our evaluation setting than the author-provided configurations when applying the scoring scheme of the competition.

Keywords: Neural Network Verification · Automated Algorithm Configuration · SMAC3 · α,β-CROWN.

1 Introduction

Despite their widespread success across various application domains, ranging from computer vision [8] to protein structure prediction [20], deep learning models remain inherently non-robust. Carefully chosen, yet imperceptible, input perturbations that cause misclassification [35] remain a major concern that limits the responsible deployment of neural networks in safety-critical domains. Thus, to provide formal guarantees on the behaviour of neural networks, *neural network verification* techniques have been developed (see, *e.g.*, [9, 12, 21, 24, 38, 39]).

Generally, these techniques are concerned with deciding whether a given input-output property, such as local robustness within norm balls around inputs, *holds* or is *violated* for a given neural network. While *incomplete* verification methods are efficient but may be unable to decide a given property due to employed over-approximations or simplifications, *complete* methods always return a definitive result but are computationally costly. In fact, it has been shown that complete verification is an $\mathcal{NP}$-complete problem, even for the simplest properties [21, 32]. Current state-of-the-art neural network verification systems rely on the *branch-and-bound* paradigm [6], a complete method that combines cheap incomplete verification methods with systematic case splitting. One representative verification system that has shown remarkable performance in recent benchmarking studies and competitions [23, 24] is α,β-CROWN [41, 39, 45]. This solver combines the branch and bound paradigm with sophisticated improvements such as the incorporation of cutting planes [46, 44], domain clipping [47] and tight bounding methods [41] that can accommodate split constraints [39]. Still, the computational burden incurred by α,β-CROWN remains substantial on challenging verification problems [23, 24].

To fully exploit the capabilities of α,β-CROWN, it is essential to carefully set the numerous hyperparameters exposed by the system. For example, anecdotal evidence suggests that partitioning the input space during branch and bound can be beneficial when the input dimensionality is low, or that invoking a MIP solver to refine intermediate layer bounds may be advantageous when the number of non-linear activations is small. However, identifying best-performing configurations typically requires considerable domain expertise and extensive manual experimentation, which makes the tuning process challenging for non-experts and practitioners. In related domains involving the solution of $\mathcal{NP}$-hard problems, such as MIP and SAT, automated algorithm configuration has been demonstrated to effectively tackle this challenge (see, *e.g.*, [15, 16, 19]). The goal of automated algorithm configuration is to identify the best-performing configuration within a predefined search space by applying optimisation techniques over a set of representative training instances. Despite this success, the potential of automated configuration methods remains largely unexplored in the context of neural network verification. In this work, we provide, for the first time, an expert-designed search space for the state-of-the-art verification system α,β-CROWN and optimise its performance within this space using the well-known automated algorithm configurator `SMAC3` [17, 26]. We assess the performance of our automated configuration procedure using the benchmarks from the regular track of the Verification of Neural Networks Competition (VNN-COMP) 2025 [23] and compare the results with the handcrafted configurations provided by the authors for their competition entry. Our experiments demonstrate that, even under conservative time budgets for the optimisation process, SMAC is able to identify configurations that generally perform *on par* with or surpass the author-provided configurations, with inferior performance occurring only on a few benchmarks and by relatively small margins. As a consequence, when adopting the scoring scheme employed in the competition, the automatically configured version of

α,β-CROWN outperforms the manually configured solver under our evaluation conditions.

In summary, our contributions are as follows:

1. We design and publish a constrained configuration space for α,β-CROWN with 63 hyperparameters (out of 278 in total), including conditional dependencies and forbidden combinations.
2. We empirically show, on all 16 benchmarks from the regular track of VNN-COMP 2025, that automatically configured α,β-CROWN matches or slightly outperforms expert-crafted configurations under realistic tuning budgets.
3. We analyse the resulting configurations, highlighting which design choices made by the automatic configuration process match those by human experts and which ones differ, and we outline how this opens up directions for further research, such as fairer tuning protocols and automated algorithm selection.

2 Background and Related Work

In the following, we discuss the relevant background of our work, covering neural network verification and automated algorithm configuration. To provide important background information on the currently prevailing best practices regarding algorithm tuning in the neural network verification community, we introduce rules and results of the annually held Verification of Neural Networks Competition (VNN-COMP).

2.1 Neural Network Verification

Generally, *neural network verification* is concerned with proving whether a given property relating the inputs and outputs of a neural network *holds* or is *violated*. More formally, given a neural network $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ with $n, m \in \mathbb{N}$, an input domain $S_{\text{in}} \subset \mathbb{R}^n$, and an output specification $S_{\text{out}} \subset \mathbb{R}^m$, the verification problem [10] refers to deciding whether

$$\forall \mathbf{x} \in S_{\text{in}} : f_\theta(\mathbf{x}) \in S_{\text{out}}. \tag{1}$$

For example, robustness to perturbations within an ℓ_∞ ball of radius ϵ around an input $\mathbf{x}_{\text{orig}}$ with correct label y_{orig} can be expressed as

$$\begin{aligned} S_{\text{in}} &= \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{x}_{\text{orig}}\|_\infty \leq \epsilon\}, \\ S_{\text{out}} &= \bigcap_{i \in [m],\, i \neq y_{\text{orig}}} \{\mathbf{z} \in \mathbb{R}^m : (e_i - e_{y_{\text{orig}}})^\top \cdot \mathbf{z} \leq 0\}, \end{aligned} \tag{2}$$

where e_i denotes the i-th standard basis vector. Thus, S_{out} is the set of outputs for which $\mathbf{z}[y_{\text{orig}}] \geq \mathbf{z}[i]$ for all $i \neq y_{\text{orig}}$, *i.e.*, ensuring that y_{orig} is the predicted class.

Algorithms that solve the verification problem can be divided into *incomplete* and *complete* approaches [24], where the former give sound results but may not

always return a solution, while the latter are guaranteed to obtain a solution given enough time. However, it has been shown that complete verification is an $\mathcal{NP}$-complete problem [21, 32]. While early complete verification approaches aimed to exploit the capabilities of highly-optimised general purpose solvers by formulating the verification problem as a MIP [38] or SMT [21] problem, they suffered from poor scalability. To improve on this, current state-of-the-art verification systems employ a Branch and Bound (BaB) approach specifically tailored to the verification problem [6] combining cheap incomplete methods that over-approximate reachable output sets with systematic case splitting to obtain a complete algorithm (see, *e.g.*, [9, 12, 39]). More precisely, if the incomplete procedure fails to provide a solution, the original verification task is decomposed into subproblems, either by partitioning the input domain or by introducing case distinctions on the activation functions. The BaB algorithm is then recursively applied to each of the resulting subproblems. While there has been a plethora of contributions aimed at accelerating the BaB procedure, ranging from the incorporation of cutting planes [44, 46] to the adaptation of the DPLL(T) framework known from SAT solving [9] and performance prediction techniques [22], the computational costs of state-of-the-art solvers renders scalability to real-world networks and problems the prevailing challenge in research on neural network verification.

2.2 Automated Algorithm Configuration

In the context of addressing $\mathcal{NP}$-hard problems such as MIP or SAT, state-of-the-art solvers typically expose a large number of tunable parameters. It is well known that the specific instantiation of these parameters can substantially affect solver performance. However, the manual configuration of such parameters generally demands considerable domain expertise as well as extensive experimentation. Thus, it is preferable to automate the configuration process. Generally, given an algorithm A with parameter configuration space Θ, a set of problem instances Π and a cost metric $m : \Theta \times \Pi \mapsto \mathbb{R}$, the goal of algorithm configuration is to obtain a configuration $\theta^* \in \Theta$ that minimises the cost metric across instances, *i.e.*

$$\theta^* \in \arg\min_{\theta \in \Theta} \sum_{\pi \in \Pi} m(\theta, \pi) \tag{3}$$

There exist several approaches to solving this optimisation problem [33], including approaches based on local search [18], performance models [17] and racing techniques [29]. Automated algorithm configuration has been demonstrated to achieve remarkable performance, for example, in the context of SAT [15, 19] and MIP [16, 27] solving, evolutionary algorithms [2] and machine learning [13, 37]. More recently, it has also been successfully applied to optimise parameters of solvers applied to a MIP formulation [38] of the neural network verification problem [25]. One widely used configurator that regularly demonstrates state-of-the-art performance [16, 28, 34] is SMAC [17, 26], a model-based configuration algorithm based on Bayesian optimisation. At runtime, SMAC trains a random forest surrogate model to predict the performance of previously unseen

configurations. SMAC samples new configurations by maximising the expected improvement [14] given the predictions and uncertainties of the surrogate using a combination of random and local search. Lastly, SMAC employs an aggressive racing mechanism to avoid running unpromising configurations on a large subset of instances by stopping poorly performing configurations early.

2.3 The Verification of Neural Networks Competition

Since 2020, an annual competition has been held with the goal to determine the state of the art in neural network verification. For this, in the Verification of Neural Networks Competition (VNN-COMP) verification systems are run on standardised, equal-cost hardware on a wide variety of benchmarks proposed by the community [5]. In the most recent iteration in 2025, tool participants were scored with regard to their performance across 16 benchmarks ranging from traditional local robustness properties on vision datasets to more complex properties such as sentence classification in NLP, certificates for control systems and robustness of image generation [23]. To determine the winner, the final score is defined as the sum of the individual benchmark scores. For each benchmark, 10 points are awarded for every correctly solved property, while each incorrect result incurs a penalty of 150 points. Lastly, each benchmark score is normalised by the maximum number of points achieved by any tool on that benchmark, *i.e.*, the highest-scoring tool receives a benchmark score of 100. Since 2021, the verification system α,β-CROWN [39, 41, 44–46] has won the competition, often by considerable margins [1, 3, 4, 23, 31]. In 2024 and 2025, the DPLL(T)-based system NeuralSAT [9] has secured second place overall [3, 23].

To assess verification systems at their optimal performance, the rules of VNN-COMP allow hyperparameter tuning on a per-benchmark basis; that is, participants may provide separate tool configurations for each benchmark, which can lead to substantial changes in solving strategies and overall behaviour. Although VNN-COMP enforces fair comparison by employing identically configured, cost-equivalent AWS hardware for the final evaluation [5, 23], it does not impose any constraints on the scope or intensity of parameter tuning conducted prior to this evaluation. Therefore, it is possible that some verification systems currently outperform others due to more effective hyperparameter tuning, while more consistent and fair tuning could lead to different competition outcomes.

2.4 Algorithm Configuration in Neural Network Verification

As mentioned previously, automated algorithm configuration techniques were successfully applied to early advancements in neural network verification based on MIP solvers [25]. However, the potential for automatically configuring current state-of-the-art verification systems based on BaB remains largely unexplored. This is mostly due to the fact that these solvers exhibit numerous parameters with complex interdependencies. For example, α,β-CROWN has over 200 potentially tunable parameters; these in turn allow for many configurations that are invalid and lead to crashes. For instance, when configuring the solver to perform

branching over the input domain within the BaB procedure, several choices of branching heuristics become inapplicable, as they are specifically designed for selecting which activation unit should be branched on rather than for partitioning the input space. Thus, the design of a configuration space that could be used in automated algorithm configuration has not succeeded to date.

Alternatively, the NeuralSAT verification system follows an orthogonal approach [9]. It does not expose any parameters to the user but sets them according to predefined heuristics based on anecdotal evidence such as enabling input splitting up to a predefined input dimension or optimising intermediate bounds using a MIP solver up to a certain number of activations in the network. Thus, NeuralSAT can be applied to novel verification benchmark tasks without requiring domain expertise and does not provide specifically crafted configurations for every benchmark in the VNN-COMP. Despite this fact, NeuralSAT achieved second place in recent renditions of the VNN-COMP [3, 23], motivating the question of how competition outcomes might change once parameter tuning is conducted in a fair and standardised fashion. A similar challenge previously arose in the context of the SAT Competition. In that setting, solvers are evaluated using only a single default configuration, with no parameter tuning permitted on a per-benchmark basis, raising the question of how solvers would perform if they were adequately tuned for each benchmark. In the Configurable SAT Solver Challenge [19], solvers were submitted together with a definition of a parameter search space, which was then used in combination with a standardised automated configuration procedure and uniform time budget to evaluate performance after fair tuning.

3 Method

As discussed above, enabling automated algorithm configuration for modern neural network verification systems is challenging, because these solvers expose a large number of parameters with complex interdependencies. Here, we consider α,β-CROWN as a representative state-of-the-art verification system and investigate how automated algorithm configuration can be effectively applied to it. Our analysis revealed that α,β-CROWN provides 278 configurable parameters. Performing algorithm configuration over the full parameter space would be computationally demanding and would most likely not yield well-performing configurations within reasonable time budgets; the large number of parameters substantially increases the size of the search space and thus the number of evaluations required to reliably identify regions of high performance. Therefore, by examining existing configurations provided by the authors for their VNN-COMP 2025 submission and analysing them, we identified a relevant subset of 63 hyperparameters that define the configuration space used in our experiments. Furthermore, through extensive experimentation and inspection of the codebase, we identified constraints that prevent infeasible parameter combinations and incorporated these constraints into the optimisation process. This allows the algorithm configurator to restrict its exploration to feasible regions of the search

space. Finally, we introduced mechanisms to exclude parts of the configuration space based on inexpensive heuristics; for instance, it is well known that input domain splitting is only beneficial when the input dimensionality is small.

3.1 Configuration Space Design

Generally, the complexity in configuring α,β-CROWN stems from the fact that the solver combines a wide variety of solving strategies and techniques. First, the system attempts quick verification through an adversarial attack and, optionally, through cheap incomplete verification using a bound propagation technique. After that, one complete verification approach is invoked and run until a specified cutoff time is reached. In the following, we explain the most important design decisions underlying the configuration space used for automated algorithm configuration. To focus on the most relevant parameters, we limited our selection to those frequently used in the VNN-COMP 2025 configurations. We assume that these parameters represent the most critical hyperparameters and that the components activated in these configurations correspond to the best-performing methods. All other parameters are left at their default values. We give an overview of the complete configuration space along with all constraints in Appendix A.

Complete Verification. The first decision to make when configuring the system is whether the verification query should be solved using a MIP formulation [38], BaB or a combination of the two, where intermediate layer bounds are refined using a MIP solver before the BaB algorithm is started. It should be noted that solving the MIP formulation may be more efficient than BaB on smaller networks, where few non-linearities need to be encoded, whereas BaB is usually preferable for larger networks.

MIP Solving. When the verification problem is addressed via the MIP formulation, we do not optimise any additional hyperparameters for complete verification. However, we set the number of parallel solvers and threads per solver to the number of available CPU cores. If intermediate bounds are optimised using a MIP solver before BaB, we additionally included the timeout for improving each intermediate layer bound as well as the percentage of the total timeout used for MIP solving as hyperparameters to be optimised.

Bound Propagation. To attempt to solve the verification problem initially using incomplete verification and within the BaB procedure, α,β-CROWN employs bound propagation techniques. We incorporated α-CROWN [41], CROWN [45], as well as the hybrid method Forward+CROWN [40] into our configuration space. Among these, α-CROWN yields the tightest bounds but incurs the highest memory consumption, Forward+CROWN is the most memory-efficient at the cost of looser bounds, and CROWN realises a compromise between tightness

of bounds and memory requirements. When α-CROWN is enabled, we included additional parameters, such as the learning rate, the number of optimisation steps and the learning rate decay used within the optimisation of α-parameters. Lastly, we always optimise whether to use `matrix` or `patches` mode for convolutions during bound propagation, where the latter is more efficient but does not support all architectures.

Branch and Bound. Regarding the BaB procedure, we added the number of domains processed in parallel to the configuration space, as this parameter strongly affects memory consumption and can help mitigate *out-of-memory* (OOM) errors. We further considered optimisation hyperparameters of the β variables in β-CROWN [39], including the learning rate and the number of optimisation iterations. In addition, the configuration space also captures whether branching is performed on the input domain or on non-linear activations. This decision determines which branching heuristics can be employed to select the next input or activation split. Specifically, *(K-)FSB* [7] and *BaBSR* [6] are applicable to activation splits, whereas *BaBSB* [6] and the brute-force *naïve* heuristic are restricted to input splits. Lastly, we also incorporated the more recent advancements *BICCOS* [46] and *Clip-and-Verify* [47] along with their method-specific settings into the configuration space. These techniques can be optionally enabled during the optimisation process to potentially accelerate the solving procedure.

Attack. Finally, we included the hyperparameters of the adversarial attack used to identify property violations in the configuration space. We support Projected Gradient Descent (PGD) [30] as well as PGD with output-diversified initialisation [36] in the search space, along with their standard hyperparameters, such as number of optimisation steps, restarts and step size. Additionally, we optimise whether the attack is executed prior to or following incomplete verification, since choosing the appropriate order can reduce computation time when the majority of properties either hold or are violated.

Search Space Reduction. Although our search space is restricted to the most frequently used hyperparameters, it still comprises 63 parameters, which remains a substantial number and can require long optimisation times to thoroughly explore the configuration space and to obtain well-performing configurations. However, several components of the search space can be eliminated using cheap heuristics. For instance, solving the MIP formulation is only feasible for smaller networks, and splitting on the input domain has only been shown to have competitive performance when the input dimensionality is low. Consequently, we provide the option to exclude certain methods from the search space based on domain knowledge or heuristic rules. Finally, rather than tuning all individual parameters of the attack, we allow optimisation over predefined attack *presets* that represent varying strengths, ranging from *weak* to *extreme* (see Appendix A, Table 5), thereby reducing the search space by six hyperparameters.

3.2 Optimisation Settings

Generally, the SMAC algorithm configurator used in our study is agnostic to the performance metric to be optimised; here, we minimise the PAR-10 score, *i.e.*, the average penalised running time in which unsolved instances are assigned a penalty equal to ten times the cutoff time. Other potential choices include a soundness loss, which penalises incorrect verification results when correct results are known *a priori*. For the optimisation, we employed the *Algorithm Configuration Façade* from `SMAC3` using its standard settings. This setup utilises the *Expected Improvement* acquisition function combined with a mixed search strategy, where random search is applied with a 50% probability alongside local search. Furthermore, SMAC employs the ROAR racing mechanism [17] and a random forest surrogate model during optimisation. Prior to starting the optimisation, the default parameter values are evaluated once per training instance to establish a baseline.

4 Empirical Evaluation

In the following, we present an empirical evaluation of our method for automatically configuring α,β-CROWN on the benchmarks of the regular track of VNN-COMP 2025 [23]. Our experiments show that, even when used with relatively modest computational budgets, SMAC consistently identifies configurations that outperform default settings and, on some benchmarks, outperform the configurations submitted by the authors for participation in the competition.

4.1 Setup of Experiments

To ensure a realistic and fair comparison, we ran the algorithm configuration procedure on the publicly available training instances of the VNN-COMP benchmarks that were used by tool authors to craft the configurations used in the competition. These instances were generated using the same procedure as the final competition instance sets, but using a different seed for the pseudo-random number generator. The final evaluation was conducted on the corresponding test instances used in the competition. During configuration, the optimisation objective was to minimise the PAR-10 score per benchmark, where each instance was subject to an individual cutoff time specified by the benchmark authors. The algorithm configuration using SMAC ran for 24 wall-clock hours per benchmark; thus, the computational demands of our procedure can be comfortably met within the three-week period between the finalisation of benchmarks and the competition submission deadline, even using a single compute node. We used the December 2025 release of α,β-CROWN at commit `6b8bbcf`, version 2.1.0 of `SMAC3` for the optimisation, and version 0.7.1 of `ConfigSpace` to define the search space. The code used to conduct our experiments is accessible in a public Git repository.[3] We executed the experiments on a compute cluster in which each

[3] `https://github.com/ADA-research/ac_abcrown`

Table 1: Number of solved, timed-out, and unknown (*i.e.*, crashed) instances per benchmark for three configurations: default (DEFAULT), automatically configured (SMAC), and the author-provided configuration (AUTHOR). The highest number of *solved* instances per benchmark is highlighted in **bold**.

	DEFAULT			SMAC (OURS)			AUTHOR		
Benchmark	Sol.	T/O	Unk.	Sol.	T/O	Unk.	Sol.	T/O	Unk.
acasxu	90	95	1	185	1	0	**186**	0	0
cersyve	4	8	0	**12**	0	0	**12**	0	0
cgan	12	2	7	19	1	1	**21**	0	0
cifar100	121	79	0	**159**	41	0	131	69	0
collins_rul_cnn	62	0	0	**62**	0	0	**62**	0	0
cora	152	28	0	152	28	0	**153**	27	0
dist_shift	68	4	0	**72**	0	0	**72**	0	0
linearizenn	27	33	0	**60**	0	0	**60**	0	0
malbeware	146	4	0	**150**	0	0	**150**	0	0
metaroom	99	1	0	**99**	1	0	**99**	1	0
nn4sys	42	0	152	184	0	10	**194**	0	0
safenlp	972	108	0	**1080**	0	0	**1080**	0	0
sat_relu	48	50	0	**98**	0	0	**98**	0	0
soundnessbench	12	38	0	49	0	1	**50**	0	0
tinyimagenet	169	31	0	**197**	3	0	178	22	0
tllverifybench	20	12	0	**32**	0	0	**32**	0	0
Total	2044	493	160	**2610**	75	12	2578	119	0

node was equipped with two Intel Xeon Platinum 8468 with 210 MB of L3 cache, four NVIDIA H100 NVL GPUs and 512 GB of RAM, running Rocky Linux 9. Here, each experiment utilised 24 CPU cores, 120 GB of RAM and one GPU. We note that this hardware setup differs from the AWS instances used during the VNN-COMP and, therefore, the configurations crafted for the competition might not exploit the full potential of the evaluation hardware used in our experiments. However, the expenses associated with running the experiments on AWS hardware were not feasible within the scope of our work presented here.

On certain benchmarks, the VNN-COMP configuration files include settings that apply modifications or simplifications to the ONNX files representing the neural networks to be verified. Because these adjustments are often essential for enabling the successful execution of α,β-CROWN and could be incorporated into preprocessing scripts prior to benchmark execution, we treat these settings as fixed during both the algorithm configuration and evaluation phases.

To improve the efficiency of the optimisation process, we excluded input splitting from the configuration space for networks with input dimensions exceeding 1 000, and we omitted verification using MIP solvers for networks with more than 2 000 000 parameters. These constitute simple heuristics that restrict the configuration space to its most relevant regions and, based on our experience, capture the best-performing methods across different benchmark types while remaining

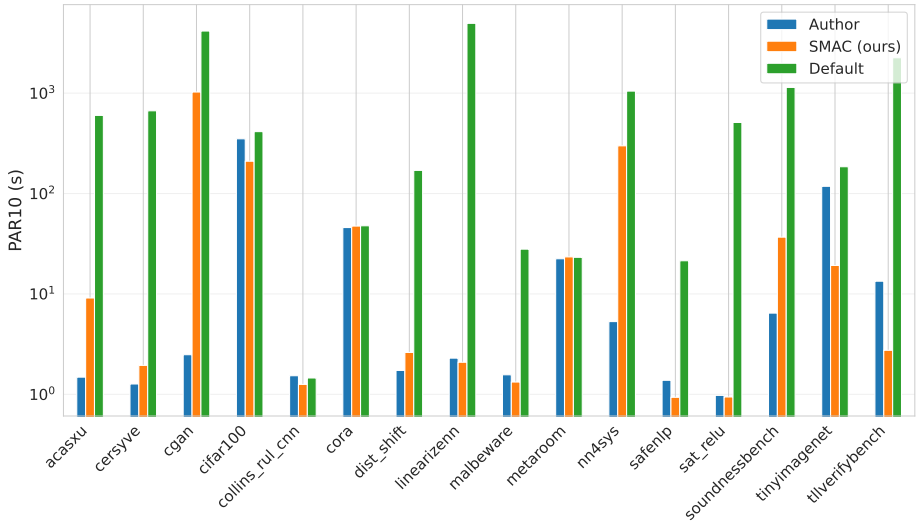


Fig. 1: PAR-10 scores (on a logarithmic scale) for the benchmarks from the regular track of VNN-COMP 2025 [23]. We compare the default configuration, the configuration provided by the authors for their entry in the competition and the automatically-tuned configuration using `SMAC3` (lower is better).

consistent with commonly used configurations. Lastly, we optimised for attack presets, *i.e.*, we did not include all hyperparameters of the adversarial attack in the search space.

We evaluated the configurations obtained using SMAC by comparing them against two baselines: the default configuration of α,β-CROWN, which uses branch and bound with activation splitting under default settings, and the configurations used by the α,β-CROWN authors in their VNN-COMP 2025 submission. Although the default configuration can be a weak baseline on some benchmarks, *e.g.*, on low-dimensional data, where input splitting with default settings may perform substantially better, it represents the configuration most readily available to practitioners without any verification expertise. We therefore included it to assess whether automated algorithm configuration can help such users obtain stronger robustness proofs without requiring domain-specific knowledge. In contrast, the author-provided configurations serve as a strong expert baseline, allowing us to assess whether automated configuration can match the performance attainable by expert tool developers.

4.2 Results

In Table 1, we report the number of solved, unsolved and crashed instances comparing the configuration found using SMAC with the default and author-provided configurations across all benchmarks of the VNN-COMP 2025 regular

track. Overall, SMAC found configurations that performed at least as well as the default configuration. This is especially pronounced on the *cersyve*, *soundnessbench*, and *nn4sys* benchmarks, where the number of solved instances increased by factors of 3.0×, 4.1×, and 4.4×, respectively. Overall, compared to the default configuration, α,β-CROWN could solve almost 600 additional instances across all benchmarks when configured using SMAC. Furthermore, for most benchmarks, the automatically configured algorithm achieved performance comparable to that of the configurations provided by the authors. On 9 of the 16 benchmarks, our configurations matched the author-provided configurations regarding the number of solved instances.

In cases where the author-provided configurations outperformed those found by SMAC, the differences were generally small. For example, the largest absolute gap in favour of the author-provided configuration occurred on the *nn4sys* benchmark, where 10 additional instances were solved, corresponding to a relative gain of 5.15%, while the largest relative discrepancy was observed on the *cgan* benchmark, where the author-provided configuration solved 9.52% more instances. However, there were also several benchmarks on which the configuration found by SMAC outperformed the author-provided configuration. The most pronounced performance gains were realised on the *CIFAR-100* and *TinyImageNet* benchmarks, where our configuration was able to solve 28 and 19 additional instances, respectively.

In Figure 1, we display the PAR-10 scores achieved by each of the investigated configurations to examine performance differences in terms of running time. Unsurprisingly, the author-provided configurations outperformed the automatically configured solver when they could solve more instances and *vice versa*. However, among the 9 benchmarks on which both configurations solved the same number of instances, the automatically configured α,β-CROWN achieved lower PAR-10 scores on 6, with particularly large gains on the *sat_relu* and *tllverifybench* benchmarks.

Using the scoring scheme employed in VNN-COMP, *i.e.*, awarding 10 points for each correctly solved property and normalising by the maximum score achieved by any tool on the same benchmark (see Section 2.3), we found that, in our evaluation, the automatically configured version of α,β-CROWN outperformed the author-configured version (see Table 2). More specifically, our configurations achieved a total score of 1582.14 points, compared to 1572.75 points obtained by the author-provided configurations. In our experiments, we employed the internal soundness check of α,β-CROWN, which ensures that adversarial examples are only returned if they satisfy the input constraints and violate the output specification, and we did not encounter any incorrect verification outcomes; instances that timed out or in which α,β-CROWN crashed were regarded as unsolved and assigned zero points. These results further underscore the effectiveness of automatic algorithm configuration and demonstrate that such techniques can have a substantial impact on the state of the art in neural network verification.

Table 2: VNN-COMP normalised scores per benchmark (100 = best configuration on that benchmark) for the default (DEFAULT), author-provided (AUTHOR), and automatically configured (SMAC) settings.

Benchmark	DEFAULT	SMAC (OURS)	AUTHOR
acasxu	48.39	99.46	**100.00**
cersyve	33.33	**100.00**	**100.00**
cgan	57.14	90.48	**100.00**
cifar100	76.10	**100.00**	82.39
collins_rul_cnn	**100.00**	**100.00**	**100.00**
cora	99.35	99.35	**100.00**
dist_shift	94.44	**100.00**	**100.00**
linearizenn	45.00	**100.00**	**100.00**
malbeware	97.33	**100.00**	**100.00**
metaroom	**100.00**	**100.00**	**100.00**
nn4sys	21.65	94.85	**100.00**
safenlp	90.00	**100.00**	**100.00**
sat_relu	48.98	**100.00**	**100.00**
soundnessbench	24.00	98.00	**100.00**
tinyimagenet	85.79	**100.00**	90.36
tllverifybench	62.50	**100.00**	**100.00**
Sum	1084.00	**1582.14**	1572.75

4.3 Analysis of Configurations

Overall, our results indicate that automated algorithm configuration is effective for state-of-the-art neural network verification systems. A closer analysis of the configurations identified by SMAC revealed that, for all benchmarks, the main solving paradigm used by the authors was recovered. In particular, SMAC correctly identified whether BaB with or without input splitting should be employed, whether the MIP problem should be solved, whether an especially strong adversarial attack is needed and whether BaB cutting planes or domain clipping should be used. This suggests that the author-provided configurations are indeed near-optimal and that SMAC is effective at identifying these highly performant settings.

However, on certain benchmarks, the performance of the author-provided configurations could not be matched. Sometimes, the VNN-COMP configurations of α,β-CROWN incorporate online heuristics to adjust the solver dynamically, *e.g.*, by setting the batch size according to the remaining available memory on the *cgan* benchmark. For the *nn4sys* benchmark, the verification system executes specialised code that reduces the number of specifications processed in parallel, based on both the parameter count of the network and the number of specifications. We note that these heuristics are only invoked on specific benchmarks and are not part of the general verification code. While these adjustments might be necessary to achieve maximum performance, such settings can naturally not be uncovered using offline algorithm configuration techniques. For the

benchmarks where the configurations found by SMAC outperformed the author-provided configurations, only minor differences were observed, suggesting that the author configurations are close to optimal. In particular, on the *CIFAR-100* and *TinyImageNet* benchmarks, SMAC selected a smaller batch size and enabled domain clipping [47] within α-CROWN, while otherwise producing a similar configuration.

5 Discussion

While our results are overall very promising, we also identified conceptual challenges in applying automated algorithm configuration to neural network verification, particularly related to over-tuning [11]. While overfitting to training instances is a well-known issue in algorithm configuration that can often be alleviated by using a sufficiently large, diverse, and representative training set, this problem is even more important in neural network verification, since performance on different types of instances is strongly impacted by different kinds of parameters. Trivially safe instances are typically resolved via incomplete verification, whereas trivially unsafe instances are handled using adversarial attacks; only the more difficult instances require complete verification. If the configurator is not exposed to a sufficient number of training examples from each of these categories, certain components of the verification system may be neglected and inadequately tuned, which in turn can degrade test performance.

Therefore, a promising direction for future work is to provide SMAC with access to instance features correlated with instance hardness, as recently employed in empirical performance prediction for neural network verification [22]. Such features could then guide the selection of training instances of varying difficulty during the configuration process. In addition, both the author-provided configurations and those identified by SMAC may be over-tuned to the respective evaluation hardware [11], *i.e.*, the AWS instances used by the authors and the compute cluster employed for SMAC in our experiments. Since the hardware environments differ, definitive conclusions about the superiority of automated tuning over careful manual tuning cannot be drawn, as the author-provided configurations might outperform ours on the AWS hardware. Nevertheless, the fact that some configurations found by SMAC outperformed the author-provided ones in our evaluation indicates that automated algorithm configuration can identify settings that better exploit the available hardware. This, in turn, enables practitioners to obtain well-performing configurations tailored to their execution environment, reducing reliance on configurations tuned on different systems.

Lastly, on many VNN-COMP benchmarks, both the author-provided and the configurations tuned using SMAC solved all instances within the given compute budget. This suggests that these benchmarks may be too easy to expose potential gains from improved configurations found by automated algorithm configuration. Our results therefore also motivate the development of more challenging benchmarks, for which better tuning can actually result in pronounced performance improvements, and their inclusion in VNN-COMP.

6 Conclusions and Future Work

In this work, we have, for the first time, successfully applied automated algorithm configuration techniques to the state-of-the-art neural network verification system α,β-CROWN. We demonstrated the effectiveness of our approach on the benchmarks of VNN-COMP 2025 [23], showing that the automatically configured solver overall outperforms the configurations determined by human experts in our experimental setting when adopting the scoring scheme of VNN-COMP. More precisely, our configurations were found to solve 28 and 19 additional instances on the *CIFAR-100* and *TinyImageNet* benchmarks, respectively, and to achieve lower PAR-10 scores on 50% of benchmarks. Furthermore, our analysis revealed that, in cases where the expert-designed configurations outperformed those found by our automated approach by the largest margins, these configurations relied on specialised online tuning mechanisms that cannot, in principle, be discovered through offline configuration.

These results were made possible by the careful design of a modular search space that includes only the most relevant parameters among the more than 250 tunable parameters exposed by α,β-CROWN. Furthermore, the modular design allows certain solving strategies to be included or excluded based on inexpensive heuristics, *e.g.*, enabling the optimisation of input domain splitting only when the input dimensionality is low, in order to reduce the size of the search space. Taken together, we believe that these contributions make it possible to leverage high-performing hyperparameter settings for α,β-CROWN even for non-expert practitioners. Lastly, we believe that this work opens several important avenues for future work. First, the proposed search space could be leveraged to identify multiple complementary configurations, which could then be integrated into an algorithm selection framework. Based on instance features, such a framework would select the most promising configuration for each instance as demonstrated previously in the contexts of SAT and MIP solving [42, 43]. Furthermore, this work demonstrates that, in principle, automated algorithm configuration is effective for neural network verification systems. To enable fair comparisons between verification tools, including equitable hyperparameter tuning, one could envision a *Configurable VNN-COMP*, inspired by the *Configurable SAT Solver Challenge* [19], in which authors provide configuration *spaces* rather than fixed configurations. These spaces could then be optimised using state-of-the-art algorithm configurators, allowing uniform and unbiased tuning and offering a fresh perspective on the state of the art in neural network verification.

Acknowledgments. The authors would like to express their sincere gratitude to Hadar Shavit for providing valuable feedback on the manuscript and for sharing his expert knowledge on algorithm configuration, and to Anna Münz for her constructive comments on the paper. Holger H. Hoos gratefully acknowledges support through an Alexander-von-Humboldt Professorship in Artificial Intelligence. Computational resources were provided by the German AI Service Center WestAI. Furthermore, the authors thank the reviewers for their valuable comments.

A Configuration Space

Table 3: Configuration space of $\alpha\beta$-CROWN. Each row lists a hyperparameter with its type, domain, default value, and the condition under which it is active. Parameters marked *const.* are fixed and never sampled. **F x** cross-references the forbidden clause in Table 4.

Parameter	Type	Domain / Choices	Def.	Condition / Note
§ 1 `general`				
`device`	const.	`cuda` / `cpu`	hw-dep.	Always active. GPU if available.
`conv_mode`	categ.	`patches`, `matrix`	`patches`	Active if `complete_verifier` ≠ `mip`.
`complete_verifier`	categ.	`bab`, `bab-refine`, `mip`	`bab`	Root switch; governs all sub-spaces.
`enable_incomplete_verification`	bool	{T, F}	T	**F7**: forbidden `False` if verifier ≠ `bab`.
§ 2 `solver`				
`batch_size`	ordinal	$\{1, 16, 64, \ldots, 65536\}$	64	Active if verifier ≠ `mip`.
`bound_prop_method`	categ.	`alpha-crown`, `crown`, `forward+crown`	`alpha-crown`	Always active.
§ 3 `solver:alpha-crown`				
`lr_alpha`	float†	$[0.001, 0.5]$	0.1	
`iteration`	int†	$[1, 500]$	100	Active if
`share_alphas`	bool	{T, F}	F	`bound_prop_method` =
`lr_decay`	float†	$[0.8, 1.0]$	0.98	`alpha-crown`.
`full_conv_alpha`	bool	{T, F}	T	
§ 4 `solver:beta-crown`				
`lr_alpha`	float†	$[0.001, 0.25]$	0.01	
`lr_beta`	float†	$[0.001, 0.5]$	0.05	Active if verifier ≠
`lr_decay`	float†	$[0.8, 1.0]$	0.98	`mip`.
`iteration`	int†	$[1, 100]$	50	
§ 5 `solver:mip`				
`refine_neuron_timeout`	int	$[1, 30]$	15	Active if verifier = `bab-refine`.
`refine_neuron_time_percentage`	float	$[0.1, 0.9]$	0.8	
`parallel_solvers`	const.	no. of cores	h.w.-dep.	Set to available number of CPU cores.
`solver_threads`	const.	no. of cores	h.w.-dep.	

Continued on next page.

Table 3 continued.

Parameter	Type	Domain / Choices	Def.	Condition / Note
§ 6 `bab`				
`pruning_in_iteration`	bool	{T, F}	T	Active if verifier ≠ `mip`.
`pruning_in_iteration_ratio`	float	$[0, 1]$	0.2	Active if `pruning_in_iteration` = T.
`interm_transfer`	bool	{T, F}	T	**F6**: forbidden `False` if verifier ≠ `bab`.
`tree_traversal`	categ.	`breadth_first`, `depth_first`	`depth_first`	Active if verifier ≠ `mip`.
§ 7 `bab:cut` & `bab:cut:biccos`				
`cut:enabled`	bool	{T, F}	F	Active if verifier = `bab`. **F8**: forbidden `True` with `fsb`/`sb`/`babsr`/`naive`.
`biccos:drop_ratio`	float	$[0, 1]$	0.5	Active if `cut:enabled` = T.
`biccos:max_infer_iter`	int	$[1, 100]$	20	
`multi_tree_branching:enabled`	bool	{T, F}	T	
`multi_tree_branching:iterations`	int	$[1, 10]$	3	Active if `cut:enabled` = T.
§ 8 `bab:branching`				
`method`	categ.	`kfsb`, `fsb`, `sb`, `kfsb-intercept-only`, `babsr`, `naive`	`kfsb`	**F1, F2, F9**: `sb`/`naive` require input-split; activation-split methods forbidden with input-split.
`candidates`	int	$[1, 20]$	3	Active if verifier ≠ `mip`.
`reduceop`	categ.	`min`, `max`	`min`	
§ 9 `bab:branching:input_split`				
`enable`	bool	{T, F}	F	**F1, F5**: forbidden `True` with activation split or verifier ≠ `bab`.
`adv_check`	categ.	$\{-1, 0\}$	0	Active if `enable` = T.
`sb_coeff_thresh`	float†	$[10^{-4}, 2]$	0.001	Active if `method` = `sb`.
§ 10 `bab:clip_n_verify`				
`clip_input_domain:enabled`	bool	{T, F}	F	**F3**: forbidden `True` if `input_split:enable` = F.
`clip_input_domain:clip_type`	categ.	`complete`, `relaxed`	`relaxed`	Active if `clip_input_domain:enabled` = T.

Continued on next page.

Table 3 continued.

Parameter	Type	Domain / Choices	Def.	Condition / Note
`clip_interm_domain: enabled`	bool	{T, F}	F	**F4**: forbidden `True` if `input_split:enable` = T.
`clip_interm_domain: clip_in_alpha_crown`	bool	{T, F}	F	Active if `clip_interm_domain: enabled` = T.
§ 11a `attack` *(preset mode,* ***tune_attack_extensive=False****)*				
`attack_preset`	ordinal	`weak` < `strong` < `super_strong` < `extreme`	`strong`	Always active.
`pgd_order`	categ.	`before`, `after`	`before`	Always active.
§ 11b `attack` *(extensive mode,* ***tune_attack_extensive=True****)*				
`pgd_steps`	int†	$[1, 1000]$	100	Always active (extensive mode).
`pgd_restarts`	int†	$[1, 10^4]$	30	
`pgd_lr_decay`	float†	$[0.5, 0.99]$	0.99	
`pgd_alpha`	float†	$[10^{-3}\varepsilon, 2\varepsilon]$	$\varepsilon/4$	Bounds depend on perturbation radius ε.
`attack_mode`	categ.	`PGD`, `diversed_PGD`	`PGD`	Always active (extensive mode).
`pgd_order`	categ.	`before`, `after`	`before`	

†Sampled on a log scale. **§ 11a** and **§ 11b** are mutually exclusive; exactly one attack subspace is instantiated at construction time. *Const.* parameters are present in the space but never sampled. All `bab:*` parameters inherit the root condition `complete_verifier` ≠ `mip` unless a stricter condition is stated.

Table 4: Forbidden clauses of the configuration space. Each row defines a pair of conditions that must *never* hold simultaneously. Cross-references match the **F x** labels in Table 3.

#	Condition A	AND Condition B (jointly forbidden)
F1	`input_split:enable` = `True`	`branching:method` ∈ {`kfsb`, `babsr`, `fsb`, `kfsb-intercept-only`}
	Activation-split methods are incompatible with input-split mode.	
F2	`branching:method` ∈ {`sb`, `naive`}	`input_split:enable` = `False`
	***`sb`/`naive`** only make sense when input-split is active.*	
F3	`input_split:enable` = `False`	`clip_input_domain:enabled` = `True`
	Input-domain clipping requires input-split.	
F4	`input_split:enable` = `True`	`clip_interm_domain:enabled` = `True`
	Input-split and intermediate-domain clipping are mutually exclusive.	
F5	`input_split:enable` = `True`	`complete_verifier` ≠ `bab`
	*Input-split is only supported with **`bab`** as the complete verifier.*	
F6	`interm_transfer` = `False`	`complete_verifier` ≠ `bab`
	***`interm_transfer`** must stay **`True`** for **`bab-refine`/`mip`**.*	
F7	`enable_incomplete_verification` = `False`	`complete_verifier` ≠ `bab`
	*Incomplete verification must remain enabled for **`bab-refine`/`mip`**.*	
F8	`cut:enabled` = `True`	`branching:method` ∈ {`fsb`, `sb`, `babsr`, `naive`}
	*BICCOS cuts are only compatible with **`kfsb`/`kfsb-intercept-only`**.*	
F9	`branching:method` ∈ {`sb`, `naive`}	*(no input-split sub-space instantiated)*
	*When input-split is globally excluded, **`sb`/`naive`** are unconditionally forbidden.*	

Table 5: Fixed parameter values for the four PGD attack presets used in the coarse-grained attack sub-space (§ 11a of Table 3). `pgd_alpha_scale`, `enable_mip_attack`, and `attack_mode` are identical across all presets and are therefore listed in the caption: `False`, `False`, and `PGD`, respectively.

Preset	pgd_steps	pgd_restarts	pgd_lr_decay	pgd_alpha
weak	10	5	0.990	0.250
strong	100	30	0.990	0.100
super_strong	200	100	0.990	0.010
extreme	1000	250	0.997	0.005

References

1. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. arXiv preprint arXiv:2109.00498 (2021)
2. Bezerra, L.C., López-Ibánez, M., Stützle, T.: Automatic component-wise design of multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation **20**(3), 403–417 (2015)
3. Brix, C., Bak, S., Johnson, T.T., Wu, H.: The fifth international verification of neural networks competition (VNN-COMP 2024): Summary and results. arXiv preprint arXiv:2412.19985 (2024)
4. Brix, C., Bak, S., Liu, C., Johnson, T.T.: The fourth international verification of neural networks competition (VNN-COMP 2023): Summary and results. arXiv preprint arXiv:2312.16760 (2023)
5. Brix, C., Müller, M.N., Bak, S., Johnson, T.T., Liu, C.: First three years of the international verification of neural networks competition (VNN-COMP). International Journal on Software Tools for Technology Transfer **25**(3), 329–339 (2023)
6. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H., Kohli, P., Kumar, M.P.: Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research **21**(42), 1–39 (2020)
7. De Palma, A., Bunel, R., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P.H., Kumar, M.P.: Improved branch and bound for neural network verification via lagrangian decomposition. arXiv preprint arXiv:2104.06718 (2021)
8. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: Proceedings of the 9th International Conference on Learning Representations (ICLR 2021). pp. 1–22 (2021)
9. Duong, H., Nguyen, T., Dwyer, M.B.: NeuralSAT: A high-performance verification tool for deep neural networks. In: Proceedings of the 37th International Conference on Computer Aided Verification (CAV 2025). pp. 409–423 (2025)
10. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI 2018). pp. 550–559 (2018)
11. Eggensperger, K., Lindauer, M., Hutter, F.: Pitfalls and best practices in algorithm configuration. Journal of Artificial Intelligence Research **64**, 861–893 (2019)
12. Ferrari, C., Mueller, M.N., Jovanović, N., Vechev, M.: Complete verification via multi-neuron relaxation guided branch-and-bound. In: Proceedings of the 10th International Conference on Learning Representations (ICLR 2022). pp. 1–15 (2022)
13. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems 28 (NeurIPS 2015). pp. 2962–2970 (2015)
14. Garnett, R.: Bayesian optimization. Cambridge University Press (2023)
15. Hutter, F., Babic, D., Hoos, H.H., Hu, A.J.: Boosting verification by automatic tuning of decision procedures. In: Proceedings of the 7th Conference on Formal Methods in Computer Aided Design (FMCAD'07). pp. 27–34 (2007)
16. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2010). pp. 186–202 (2010)

17. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION'05). pp. 507–523 (2011)
18. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research **36**, 267–306 (2009)
19. Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H., Leyton-Brown, K.: The configurable SAT solver challenge (CSSC). Artificial Intelligence **243**, 1–25 (2017)
20. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al.: Highly accurate protein structure prediction with AlphaFold. Nature **596**(7873), 583–589 (2021)
21. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017). pp. 97–117 (2017)
22. Kaulen, K., König, M., Hoos, H.H.: Dynamic algorithm termination for branch-and-bound-based neural network verification. In: Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI-25). pp. 27356–27364 (2025)
23. Kaulen, K., Ladner, T., Bak, S., Brix, C., Duong, H., Flinkow, T., Johnson, T.T., Koller, L., Manino, E., Nguyen, T.H., et al.: The 6th international verification of neural networks competition (VNN-COMP 2025): Summary and results. arXiv preprint arXiv:2512.19007 (2025)
24. König, M., Bosman, A.W., Hoos, H.H., van Rijn, J.N.: Critically assessing the state of the art in neural network verification. Journal of Machine Learning Research **25**(12), 1–53 (2024)
25. König, M., Hoos, H.H., Rijn, J.N.v.: Speeding up neural network robustness verification via algorithm configuration and an optimised mixed integer linear programming solver portfolio. Machine Learning **111**(12), 4565–4584 (2022)
26. Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., Hutter, F.: SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. Journal of Machine Learning Research **23**(54), 1–9 (2022)
27. López-Ibáñez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. European Journal of Operational Research **235**(3), 569–582 (2014)
28. Luo, C., Hoos, H.H., Cai, S.: PbO-CCSAT: boosting local search for satisfiability using programming by optimisation. In: Proceedings of the 16th International Conference on Parallel Problem Solving from Nature (PPSN XVI). pp. 373–389 (2020)
29. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
30. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: Proceedings of 6th International Conference on Learning Representations (ICLR 2018). pp. 1–23 (2018)
31. Müller, M.N., Brix, C., Bak, S., Liu, C., Johnson, T.T.: The third international verification of neural networks competition (VNN-COMP 2022): Summary and results. arXiv preprint arXiv:2212.10376 (2022)
32. Sälzer, M., Lange, M.: Reachability is NP-complete even for the simplest neural networks. In: Proceedings of the 15th International Conference on Reachability Problems (RP 2021). pp. 149–164 (2021)

33. Schede, E., Brandt, J., Tornede, A., Wever, M., Bengs, V., Hüllermeier, E., Tierney, K.: A survey of methods for automated algorithm configuration. Journal of Artificial Intelligence Research **75**, 425–487 (2022)
34. Seipp, J., Sievers, S., Helmert, M., Hutter, F.: Automatic configuration of sequential planning portfolios. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15). pp. 3364–3370 (2015)
35. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014). pp. 1–10 (2014)
36. Tashiro, Y., Song, Y., Ermon, S.: Diversity can be transferred: Output diversification for white- and black-box attacks. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020). pp. 1–25 (2020)
37. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2013). pp. 847–855 (2013)
38. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: Proceedings of the 7th International Conference on Learning Representations (ICLR 2019). pp. 1–21 (2019)
39. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Advances in Neural Information Processing Systems 34 (NeurIPS 2021). pp. 29909–29921 (2021)
40. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.: Automatic perturbation analysis for scalable certified robustness and beyond. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020). pp. 1–13 (2020)
41. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: Proceedings of the 9th International Conference on Learning Representations (ICLR 2021). pp. 1–15 (2021)
42. Xu, L., Hoos, H.H., Leyton-Brown, K.: Hydra: automatically configuring algorithms for portfolio-based selection. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10). pp. 210–216 (2010)
43. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Hydra-MIP: automated algorithm configuration and selection for mixed integer programming. In: Proceedings of the 18th RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA-2011). pp. 16–30 (2011)
44. Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C.J., Kolter, J.Z.: General cutting planes for bound-propagation-based neural network verification. Advances in Neural Information Processing Systems 35 (NeurIPS 2022) pp. 1656–1670 (2022)
45. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems 31 (NeurIPS 2018). pp. 4944–4953 (2018)
46. Zhou, D., Brix, C., Hanasusanto, G.A., Zhang, H.: Scalable neural network verification with branch-and-bound inferred cutting planes. Advances in Neural Information Processing Systems 38 (NeurIPS 2024) pp. 29324–29353 (2024)
47. Zhou, D., Chavez, J., Chen, H., Hanasusanto, G.A., Zhang, H.: Clip-and-Verify: Linear constraint-driven domain clipping for accelerating neural network verifica-

tion. In: Advances in Neural Information Processing Systems 39 (NeurIPS 2025). pp. 1–41 (2025)