



UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

Verificarea Rețelelor Neuronale folosind Alpha-Beta CROWN și NeuralSAT pentru Benchmark-ul Acas XU

Bordea Liviu Valentin
Ciurdea Roberta Carla
Bursea Adina Mădălina

Conf. Dr. Mădălina Erașcu

Abstract

Neural networks are being applied to a growing variety of applications, including safetycritical domains like autonomous vehicles and aircraft, due to their ability to approximate complex functions from limited datasets and adapt by continuing to learn from real-world data after deployment.

This report is focused on the installation of the two tools: alpha-beta-CROWN and NeuralSAT, which are applied to the Acas XU benchmark.

The structure of the report is based on 4 chapters. The first chapter presents the description of the chosen data set, Acas XU, the next one includes the installation of the 2 tools: alpha-beta-CROWN, NeuralSAT, and the last 2 chapters present the running of alpha-beta-CROWN, NeuralSAT on Acas XU and the final conclusions together with the problems encountered on the entire configuration of the chosen dataset.

1 Introducere

Verificarea DNN (Deep Neural Network) este un proces de confirmare a faptului că, pentru fiecare posibilă intrare, ieșirea rețelei neurale satisface proprietățile dorite. Cu alte cuvinte, este vorba despre verificarea relațiilor dintre intrarea și ieșirea rețelei neurale pentru a vedea dacă proprietățile specifice dintre ele sunt respectate.

VNN-COMP reprezintă Competiția de Verificare a Rețelelor Neurale, care s-a desfășurat anual timp de trei ani și a avut numeroși cercetători activi în acest domeniu ca participanți.

Din cea mai recentă ediție a VNN-COMP, am selectat doua instrumente care au obținut cele mai bune performanțe, au participat la cele mai multe benchmark-uri și au furnizat documentație suficientă pentru raportul nostru. Ca și benchmark, am ales AcasXU deoarece este unul dintre cele mai complexe dataset-urile, considerându-l cel mai potrivit pentru acest proiect.

Într-un sistem critic pentru securitate precum ACAS Xu, un atacator poate exploata cu ușurință o situație limită gestionată incorect pentru a cauza daune semnificative, punând în pericol mii de vieți. Metodele existente pentru testarea rețelelor neurale în fața situațiilor limită se concentrează pe identificarea exemplorilor adversare [1].

2 Descrierea Dataset-ului

ACAS XU[2] (Automated Collision Avoidance System) este un sistem de evitare a coliziunilor aer-la-aer proiectat pentru aeronave fără pilot, care emite sfaturi privind virajele orizontale pentru a evita o aeronavă intruză. Datorită utilizării unei tabele de căutare mari în proiectare, s-a propus o comprimare a politicii cu ajutorul unei rețele neurale.

Funcționalitatea Rețelei. Sistemul ACAS Xu mapează variabilele de intrare la recomandările de acțiune. Fiecare recomandare este asignată un scor, cu cel mai mic scor corespunzător celei mai bune acțiuni. Starea de intrare este compusă din șapte dimensiuni (prezentate în Figure 1) care reprezintă informațiile determinate din măsurătorile senzorilor:

- (i) ρ : Distanța de la propria navă la intrus.
- (ii) θ : Unghiul față de intrus în raport cu direcția de îndreptare a propriei nave.
- (iii) ψ : Unghiul de îndreptare al intrusului în raport cu direcția de îndreptare a propriei nave.
- (iv) v_{own} : Viteza propriei nave.
- (v) v_{int} : Viteza intrusului.

Există cinci ieșiri care reprezintă diferitele recomandări orizontale pe care le poate primi propria navă: Fără Conflict (COC), dreapta slabă, dreapta puternică, stânga slabă sau stânga puternică. Slab și puternic înseamnă rate de îndreptare de $1.5^\circ/s$ și $3.0^\circ/s$, respectiv.

Sistemul de evitare a coliziunilor cu aeronave fără pilot X (ACAS Xu) folosește rețele neuronale pentru a anticipa cele mai bune acțiuni optime în funcție de locația și viteza avioanelor atacatoare din apropiere. A fost testat cu succes de NASA și FAA și este în program pentru a fi instalat la peste 30.000 de aeronave de pasageri și cargo la nivel mondial, precum și în flotele Marinei SUA. [4]

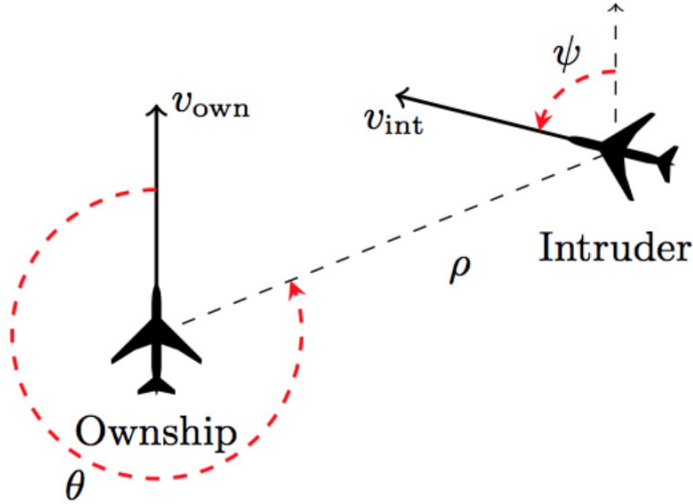


Figure 1: Sistemul ACAS Xu (adaptat din (Katz et al. 2017)) [3]

Utilitate. Dosarul de unde se poate găsi benchmark-ul ACAS Xu, conține fișierele .onnx și .vnnlib utilizate pentru această categorie, și poate fi descărcat de aici.

Fișierul `acasxu_instances.csv` conține lista completă a instanțelor de benchmark, câte una pe linie: `onnx_file`, `vnn_lib_file`, `timeout_secs`. Fișierele .vnnlib și .csv au fost create cu ajutorul scriptului `generate.py` inclus. Fișierul cu extensia ".onnx" reprezintă modelul neuronal, iar cel cu extensia ".vnnlib" specificațiile.

Rețele (onnx): ACASXu constă în zece proprietăți definite pentru 45 de rețele neurale utilizate pentru a emite avertismente de viraj aeronautic pentru evitarea coliziunilor. Rețelele neurale au 300 de neuroni aranjați în 6 straturi, cu funcții de activare ReLU. Există cinci intrări corespunzătoare stării aeronavelor și cinci ieșiri ale rețelei, unde ieșirea minimă este utilizată ca avertisment de viraj produs în cele din urmă de sistem.

Specificatii (vnnlib): Sunt utilizate cele zece proprietăți originale (vnnlib), unde proprietățile 1-4 sunt verificate pe toate cele 45 de rețele, așa cum s-a făcut ulterior de către autorii originali. Proprietățile 5-10 sunt verificate pe o singură rețea. Prin urmare, numărul total de benchmark-uri este de 186.

Ca standard pentru VNN-COMP 2022, instrumentele sunt compatibile cu modelele de rețele neurale în formatul Open Neural Network Exchange (ONNX), care pot fi convertite ușor din alte formate de fișiere ale altor cadre de învățare automată. Pentru specificații, se utilizează formatul VNNLIB, creat în mod specific pentru competiție. Acest format de specificații constă în declarații de variabile și afirmații (assertion statements), așa cum este prezentat în Figure 2.

(declare-const X_0 Real)	(assert (>= X_0 20.000000))	(assert (or
(declare-const X_1 Real)	(assert (<= X_0 25.000000))	(and (<= Y_2 Y_0))
(declare-const X_2 Real)	(assert (>= X_1 23.000000))	(and (<= Y_2 Y_1))
(declare-const Y_0 Real)	(assert (<= X_1 27.000000))))
(declare-const Y_1 Real)	(assert (>= X_2 8.000000))	
(declare-const Y_2 Real)	(assert (<= X_2 21.000000))	

Figure 2: Formatul Standard al Fișierelor VNNLIB

3 Instalarea tool-urilor

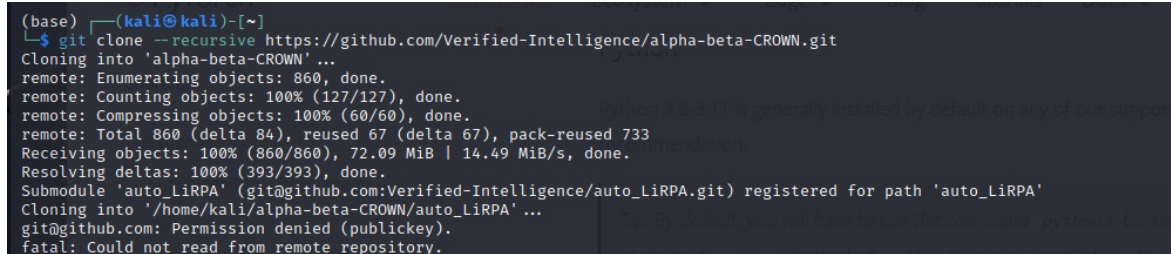
3.1 Alpha Beta-CROWN

Alpha-beta-CROWN[6] este un vericator de rețele neurale bazat pe un cadru eficient de propagare a limitelor liniare și pe tehnica de branch and bound. Poate fi accelerat eficient pe unități de procesare grafică (GPU) și poate scala către rețele convoluționale relativ mari.

Pentru a instala Alpha Beta-CROWN, am folosit miniconda3 și sistemul de operare Linux Kali.

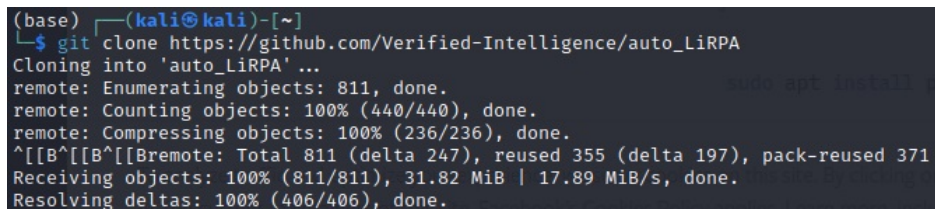
Dupa instalarea miniconda [7] și restartarea terminalului, a fost nevoie de clonarea folderului "alpha-beta-CROWN"[8] (Figure 3) și "auto_LIRPA"[9] (Figure 4) de pe github folosind instrucțiunile:

```
1 git clone --recursive https://github.com/Verified-Intelligence/alpha-  
  beta-CROWN.git  
2 git clone https://github.com/Verified-Intelligence/auto_LiRPA
```



```
(base) (kali@kali)-[~]  
└─$ git clone --recursive https://github.com/Verified-Intelligence/alpha-beta-CROWN.git  
Cloning into 'alpha-beta-CROWN' ...  
remote: Enumerating objects: 860, done.  
remote: Counting objects: 100% (127/127), done.  
remote: Compressing objects: 100% (60/60), done.  
remote: Total 860 (delta 84), reused 67 (delta 67), pack-reused 733  
Receiving objects: 100% (860/860), 72.09 MiB | 14.49 MiB/s, done.  
Resolving deltas: 100% (393/393), done.  
Submodule 'auto_LiRPA' (git@github.com:Verified-Intelligence/auto_LiRPA.git) registered for path 'auto_LiRPA'  
Cloning into '/home/kali/alpha-beta-CROWN/auto_LiRPA' ...  
git@github.com: Permission denied (publickey).  
fatal: Could not read from remote repository.
```

Figure 3: Clonarea Alpha-beta CROWN

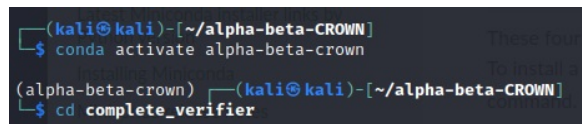


```
(base) (kali@kali)-[~]  
└─$ git clone https://github.com/Verified-Intelligence/auto_LiRPA  
Cloning into 'auto_LiRPA' ...  
remote: Enumerating objects: 811, done.  
remote: Counting objects: 100% (440/440), done.  
remote: Compressing objects: 100% (236/236), done.  
remote: Total 811 (delta 247), reused 355 (delta 197), pack-reused 371  
Receiving objects: 100% (811/811), 31.82 MiB | 17.89 MiB/s, done.  
Resolving deltas: 100% (406/406), done.
```

Figure 4: Clonarea Auto Lirpa

Am creat environmentul (Figure 6), și am activat alpha-beta-crown (Figure 5) folosind instrucțiunile:

```
1 conda env create -f complete\_verifier/environment.yaml --name alpha  
  beta-crown  
2 conda activate alpha-beta-crown
```



```
(kali@kali)-[~/alpha-beta-CROWN]  
└─$ conda activate alpha-beta-crown  
(alpha-beta-crown) (kali@kali)-[~/alpha-beta-CROWN]  
└─$ cd complete_verifier
```

Figure 5: Activarea tool-ului

```

(kali@kali)-[~]
└─$ cd /home/kali/alpha-beta-CROWN
(kali@kali)-[/home/kali/alpha-beta-CROWN]
└─$ conda env create -f complete_verifier/environment.yaml --name alpha-beta-crown
Channels:
 - gurobi
 - pytorch
 - nvidia
 - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

Downloading and Extracting Packages:

```

Figure 6: Crearea environmentului

Dupa activarea "alpha-beta-CROWN", a fost nevoie de instalarea unor pachete pentru a putea rula benchmark-ul. Aceste pachete sunt: onnx2pytorch, onnxruntime, onnxoptimizer, skl2onnx, torch, torchvision, pe care le-am instalat folosind urmatoarele instructiuni:

```

1 pip install git+https://github.com/KaidiXu/onnx2pytorch.git
2 pip install onnxruntime==1.16.3
3 pip install onnxoptimizer==0.3.13
4 pip install skl2onnx==1.15.0
5 pip install torchvision==0.15.1
6 pip install torch==1.13.0

```

3.2 NeuralSAT

NeuralSAT[10] este o unealtă de verificare a rețelelor neurale profunde (DNN - Deep Neural Network). Integrează abordarea DPLL(T)[11] folosită în mod obișnuit în rezolvarea SMT cu un solver de teorie specializat pentru raționamentul DNN. NeuralSAT valorifică unitățile de procesare multistrat (multicores) și unitatea de procesare grafică (GPU) pentru eficiență și poate scala pentru rețele cu milioane de parametri.

Pentru a instala NeuralSAT, am folosit miniconda3 și sistemul de operare Linux Kali.

Dupa instalarea miniconda și restartarea terminalului, a fost nevoie de clonarea folderului "neuralsat"[12] (Figure 7) de pe github folosind instrucțiunea:

```

1 git clone --recursive https://github.com/dynaroars/neuralsat.git

```

```

(base) (kali@kali)-[~]
└─$ git clone --recursive https://github.com/dynaroars/neuralsat.git
Cloning into 'neuralsat'...
remote: Enumerating objects: 10054, done.
remote: Counting objects: 100% (1384/1384), done.
remote: Compressing objects: 100% (459/459), done.
remote: Total 10054 (delta 937), reused 1362 (delta 918), pack-reused 8670
Receiving objects: 100% (10054/10054), 201.84 MiB | 5.26 MiB/s, done.
Resolving deltas: 100% (6218/6218), done.

```

Figure 7: Clonarea NeuralSAT

Am creat environmentul (Figure 8), și am activat neuralsat (Figure 9) folosind instrucțiunile:

```

1 conda env create -f env.yaml
2 conda activate neuralsat

```

```
(base) (kali@kali)~[~]
└─$ cd /home/kali/neuralsat/

(base) (kali@kali)~[~/neuralsat]
└─$ conda env create -f env.yaml
Channels:
- gurobi
- pytorch
- nvidia
- defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

Downloading and Extracting Packages:
```

Figure 8: Crearea environmentului

```
done
#
# To activate this environment, use
#
#   $ conda activate neuralsat
#
# To deactivate an active environment, use
#
#   $ conda deactivate

(base) (kali@kali)~[~/neuralsat]
└─$ conda activate neuralsat

(neuralsat) (kali@kali)~[~/neuralsat]
└─$
```

Figure 9: Activarea tool-ului

4 Rularea tool-urilor pentru dataset-ul ales

Pentru a rula AcasXu, am descărcat arhiva care conține toate benchmark-urile din 2023 [13]. După descărcare, am dezarhivat folderul "acasxu" și componentele acestuia. Am creat un nou folder cu denumirea "vnncomp2023_benchmarks" în care am creat folderul "benchmarks", unde am adăugat folderul "acasxu" dezarhivat și descărcat de pe github.

Din cauza unor erori, a fost necesară adăugarea unei instrucțiuni în fișierul de configurare al benchmark-ului. Instrucțiunea este "device: cpu", astfel încât pytorch să utilizeze CPU în loc de GPU.

4.1 Tool 1: Alpha-Beta CROWN

Pentru a aplica benchmark-ul și a primi rezultatele am folosit următoarea instrucțiune pe care am rulat-o în folderul "complete_verifier" aflat în "alpha-beta-CROWN":

```
1 python abcrown.py --config exp\_configs/vnncomp23/acasxu.yaml
```

Ca și rezultate, acestea pot fi vizualizate în fișierul "rezultat_abcrown.txt" [14], dar și în tabelul "results_ABCROWN.csv"[15] unde se pot observa diferențele dintre rezultatele de la competiția din 2023 și rezultatele la care am reușit să ajungem noi. Sumarul rezultatului se poate observa în Figure 10, iar în Figure 11 am evidențiat 3 tipuri de rezultate din fișierul "results_ABCROWN.csv".

```
##### Summary #####
Final verified acc: 60.215053763440864% (total 186 examples)
Problem instances count: 186 , total verified (safe/unsat): 112 , total falsified (unsafe/sat): 47 , timeout: 27
mean time for ALL instances (total 186):88.3625211687591, max time: 3492.9951119422913
mean time for verified SAFE instances(total 112): 16.101693002241, max time: 75.69043254852295
mean time for verified (SAFE + UNSAFE) instances (total 159): 12.194689128383901, max time: [26.13232421875, 22.13
mean time for verified SAFE + TIMEOUT instances (total 139): 117.26520766800256, max time: [466.40591263771057, 47
mean time for verified UNSAFE instances (total 47): 2.8843820247244327, max time: 58.81425142288208
safe (total 112), index: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 18, 19, 20, 21, 25, 26, 28, 29, 30, 36, 37
unknown (total 27), index: [14, 15, 16, 17, 22, 23, 24, 27, 31, 32, 33, 34, 35, 40, 41, 42, 43, 45, 52, 65, 73, 90
unsafe-bab (total 5), index: [46, 47, 49, 50, 83]
unsafe-pgd (total 42), index: [48, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68, 69, 70, 71, 72, 74, 75,
```

Figure 10: Sumar al rezultatului

Programul a returnat un total de 47 de cazuri satisfiabile (un-safe), 112 nesatisfiabile (safe) dar și 27 unknown (timeout). Figure 12 reprezintă procentul pe care fiecare tip de rezultat îl reprezintă din totalul de 186 de instanțe.

ONNX	VNNLIB	Competiton results	Alpha-beta CROWN-2024
ACASXU_run2a_5_9_batch_2000.onnx	prop_1.vnnlib	24.29692 unsat	41.783 unsat
ACASXU_run2a_1_1_batch_2000.onnx	prop_2.vnnlib	24.47705 unsat	454.9364 timeout
ACASXU_run2a_1_2_batch_2000.onnx	prop_2.vnnlib	22.65554 sat	2.2997 sat

Figure 11: Exemple din fișierul results_ABCROWN.csv

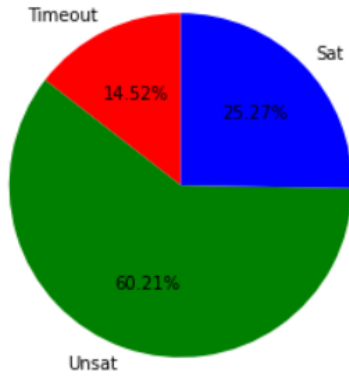


Figure 12: Diagrama Rezultatelor

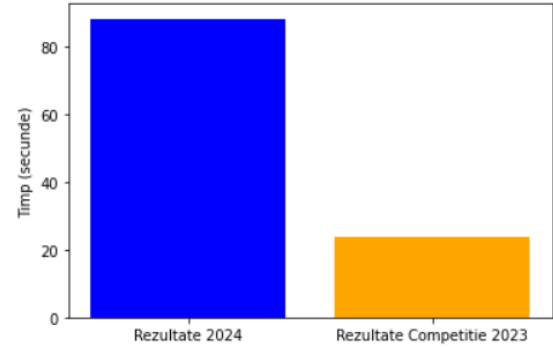


Figure 13: Comparare Timp Mediu

Satisfiabilitatea a fost aceeași ca și în rezultatele din 2023 pentru rezultatele care au fost verificate cu succes (sat, unsat), numai timpul de verificare a fost diferit într-o mare măsură. Timpul de verificare pentru rezultatele satisfiabile a fost considerabil de mic față de cele nesatisfiabile, deși în competiția trecută, timpii au fost asemanatori. De asemenea, timpul mediu de verificare pe care am reușit să îl obținem este cu aproximativ 269.86% mai mare decât cel obținut în competiția din 2023 (Figure 13). Programul a afișat 27 de rezultate cu raspunsul "timeout" deoarece nu a reușit să ajungă la un rezultat, iar timpul de verificare a expirat. Am dedus că motivul pentru care timpul de verificare este considerabil mai mare și au fost mai multe rezultate de tipul "timeout" comparativ cu 2023, este incapacitatea dispozitivului de pe care am rulat instrucțiunile (memorie insuficientă etc).

Ca și probleme întâmpinate în rularea dataset-ului cu tool-ul alpha-beta-crown au fost două situații în care în timpul rulării, programul s-a oprit cu secvența "killed" la diferite iterații probabil din cauza memoriei insuficiente (Figure 14). O altă problemă și cea mai neasteptată a fost atunci când dispozitivul s-a închis brusc și a afișat instrucțiuni de recuperare (Figure 15).

```

Iteration 16
Adv attack time: 2.7865s
Batch size: 1000
zsh: killed python abcrown.py --config exp_configs/vnncomp23/acasxu.yaml
(alpha-beta-crown) (root@kali) [/home/./Desktop/workspace/alpha-beta-CROWN]

```

Figure 14: "Killed"

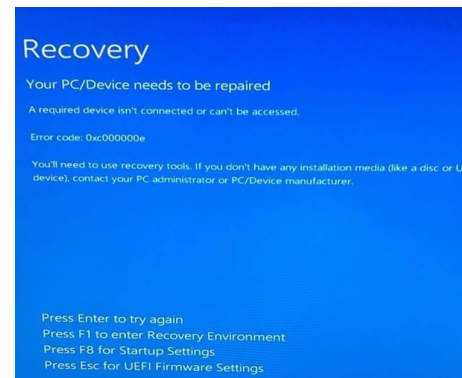


Figure 15: "Recovery"

4.2 Tool 2: NeuralSAT

Pentru a aplica benchmark-ul și a primi rezultatele am folosit următoarea instrucțiune pe care am rulat-o în folderul "neuralsat-pt201" aflat în "neuralsat":

```
1 python3 neuralsat.py
```

Pentru acest tool am folosit un program în limbajul Python personalizat deoarece NeuralSAT acceptă doar instrucțiuni de forma:

```
1 python3 main.py --net "example/onnx/mnistfc-medium-net-554.onnx" --  
spec "example/vnnlib/test.vnnlib"
```

Pentru a rula datasetul în environment-ul neuralsat, sunt necesare 2 argumente: "--net" care va fi urmat de fișierul cu extensia ".onnx", și "--spec" urmat de fișierul cu extensia ".vnnlib". Cele 2 argumente sunt cruciale pentru acest environment, iar ca soluție pentru a nu fi nevoie de executarea pe rând a 186 de instrucțiuni, am creat fișierul "neuralsat.py" care are rolul de a automatiza procesul. Acesta execută comenzile pentru fiecare combinație de fișiere, le afișează în terminal și le adaugă într-un fișier pentru a putea fi mai accesibile. Am folosit librăria "tqdm" pentru a afișa progresul parcurs din momentul începerii executiei unei combinații de fișiere până la afișarea rezultatului aferent acesteia (Figura 16). De asemenea, am implementat și un tracker pentru a gestiona numărul de combinații care au fost executate. (Figure 17).

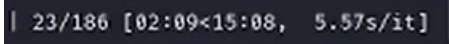


Figure 16: Bara de Progress



Figure 17: Tracker-ul Execuțiilor

Ca și rezultate, acestea pot fi vizualizate în fișierul "rezultat_neuralsat.txt" [16], dar și în tabelul "results_NeuralSat.csv" [17], unde se pot observa diferențele dintre rezultatele de la competiția din 2023 și rezultatele la care am reușit să ajungem noi. În Figura 18 am evidențiat 3 tipuri de rezultate din fișierul "results_NeuralSat.csv".

ONNX	VNNLIB	Competiton results 2023		NeuralSat 2024	
ACASXU_run2a_5_9_batch_2000.onnx	prop_2.vnnlib	3.67981	sat	0.6832	sat
ACASXU_run2a_1_1_batch_2000.onnx	prop_3.vnnlib	11.51888	unsat	>120	timeout
ACASXU_run2a_1_4_batch_2000.onnx	prop_3.vnnlib	6.65446	unsat	65.4989	unsat

Figure 18: Exemple din fisierul results_NeuralSat.csv

Programul a returnat un total de 46 de cazuri satisfiabile (un-safe), 120 nesatisfiabile (safe), dar și 20 unknown (timeout). Figura 19 reprezintă procentul pe care fiecare tip de rezultat îl reprezintă din totalul de 186 de instanțe.

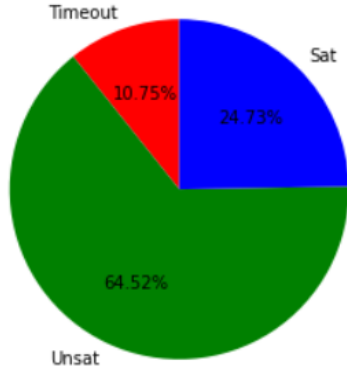


Figure 19: Diagrama Rezultatelor

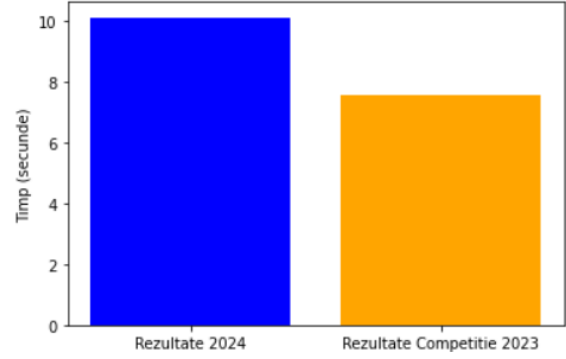


Figure 20: Comparare Timp Mediu

Satisfiabilitatea a fost aceeași ca și în rezultatele din 2023 pentru rezultatele care au fost verificate cu succes (sat, unsat), numai timpul de verificare a fost diferit într-o mare măsură. Timpul de verificare pentru rezultatele satisfiabale a fost considerabil de mic față de cele nesatisfiabale, dar o diferență asemănătoare a fost prezentă și în competiția trecută. De asemenea, timpul mediu de verificare pe care am reușit să-l obținem este cu aproximativ 33.39% mai mare decât cel obținut în competiția din 2023 (Figura 20), dar de această dată am oprit timeout-ul la 120 de secunde. Programul a afișat 20 de rezultate cu răspunsul "timeout" deoarece nu a reușit să ajungă la un rezultat, iar timpul de verificare a expirat. Am încercat reverificarea pe rând a fișierelor cu rezultat "unknown", dar după numeroase minute de așteptare, programul a returnat rezultatul "killed" (Figure 21).

```
INFO 09:21:40 [Input splitting] Iteration: 774 Remaining: 111 Visited: 222 Bound: -0.1
623 Time elapsed: 255.71 Iteration elapsed: 16.88
INFO 09:22:06 [Input splitting] Iteration: 775 Remaining: 202 Visited: 333 Bound: -0.1
349 Time elapsed: 281.04 Iteration elapsed: 25.33
zsh: killed python main.py --net acasxu/onnx/ACASXu_run2a_3_3_batch_2000.onnx --spec
```

Figure 21: Eroare "killed"

5 Sumar al Rezultatelor

Prin furnizarea unui fișier model ONNX și a unui fișier de specificații VNNLIB fiecărui instrument, rezultatul indică dacă există un contraexemplu, adică o situație în care toate condițiile din specificație sunt adevărate dat fiind modelul. Dacă există un contraexemplu, rezultatul este notat cu SAT (Satisfiabil). În caz contrar, este notat cu UNSAT (Nesatisfiabil). În unele situații, totuși, un instrument poate să nu poată să tragă o concluzie pentru verificare și va furniza ca rezultat UNKNOWN. În plus, deoarece instrumentele sunt concepute pentru aceste competiții în care scorurile depind și de timpul de execuție, dacă un instrument nu poate finaliza verificarea într-un interval de timp specificat, algoritmul este întrerupt și va fi furnizat rezultatul TIMEOUT.

În cazul experimentului pe care l-am efectuat, au existat 47 de contraexemple atunci când am aplicat α - β -CROWN asupra datasetului AcasXu și respectiv 46 la aplicarea tool-ului NeuralSat. În competiția de anul trecut, timpul maxim de verificare

a fost setat la 116 secunde pentru a nu depăși totalul de 6 ore de rulare, iar pentru cele 2 tool-uri și datasetul pe care le-am ales, au primit doar un rezultat "timeout" pentru NeuralSat. În schimb, în urma rezultatelor la care am reușit să ajungem, am identificat 27 de ieșiri "timeout" pentru α - β -CROWN, respectiv 20 pentru NeuralSAT. Din acest fapt am dedus că instrumentele nu au reușit să tragă o concluzie pentru verificare în timp util. Timpul mediu rezultat în urma execuției celor 2 tool-uri este considerabil de diferit de la un tool la altul. Acest lucru este datorat faptului că pentru α - β -CROWN timpul este compus și din secunde acumulate în urma rezultatelor 'unknown', pe când la NeuralSat acest tip de rezultate aveau un timp nedefinit, ajungându-se la eroarea 'killed' despre care am discutat mai sus. Pentru a evita stricarea programului, am setat limita de secunde la 120 și am adăugat în fișierul 'neuralsat.py' comenzi pentru a trece la următoarea combinație dacă timpul de verificare a depășit limita impusă de noi.

Tabelul "Table 1" conține rezumatul rezultatelor în urma rulării celor 2 tool-uri asupra dataset-ului AcasXu.

Tool	Total	SAT	UNSAT	Timeout	Timpul mediu
α - β -CROWN	186	47	112	27	88.362
neuralSAT	186	46	120	20	10.10

Table 1: Rezultate

Pentru a calcula scorul am folosit următoarele punctaje extrase din regulile competiției:

- **Rezultat Nesatisfiabil:** 10 puncte;
- **Rezultat Satisfiabil:** 10 puncte;
- **Rezultat Incorect:** -150 puncte.

În Tabelul 2 putem identifica următoarele: Verified (Unsat), Falsified (Sat), Fastest, Penalty (Numărul de diferențe față de competiția trecută), Score (Punctajul obținut din adunarea tuturor rezultatelor sat & unsat înmulțite cu 10. Din rezultat se scade numărul penalităților înmulțit cu 150) și Percent (Cât la sută din maximul de 1860 de puncte s-a obținut).

#	Tool	Verified	Falsified	Fastest	Penalty	Score	Percent
1	α - β -CROWN	112	47	0	27	-2460	0%
2	neuralSAT	120	46	0	19	-1190	0%

Table 2: Benchmark 2024-acasxu

6 Concluzii

În concluzie, scopul acestui proiect a fost atins deoarece am reușit instalarea și antrenarea celor două tool-uri (α - β -CROWN & neuralSat) asupra datasetului AcasXU. Rezultatele nu au fost conform așteptărilor deoarece au existat numeroase penalități care au dus la un scor negativ.

References

- [1] “Reluplex: An efficient smt solver for verifying deep neural networks.” [Online]. Available: https://www.researchgate.net/figure/Geometry-for-ACAS-Xu-Horizontal-Logic-Table_fig2_313394663
- [2] “An introduction to acas xu and the challenges ahead.” [Online]. Available: <https://enac.hal.science/hal-01638049/file/acasxu.pdf>
- [3] “Art: Abstraction refinement-guided training for provably correct neural networks.” [Online]. Available: https://www.researchgate.net/figure/The-ACAS-Xu-System-adapted-from-Katz-et-al-2017_fig1_334695061
- [4] M. Marston and G. Baca, “Acas-xu initial self-separation flight tests,” *NASA Technical Reports Server*, 2015. [Online]. Available: <https://ntrs.nasa.gov/api/citations/20150008347/downloads/20150008347.pdf>
- [5] “Formal verification of neural networks.” [Online]. Available: https://cds.cern.ch/record/2867415/files/NNVerif_CERN_Report.pdf
- [6] “Alpha-beta-crown: Neural network verifier.” [Online]. Available: <https://medium.com/@vishalisrinivasan97/alpha-beta-crown-neural-network-verifier-425281f29712>
- [7] “Quick command line install miniconda3.” [Online]. Available: <https://docs.conda.io/projects/miniconda/en/latest/>
- [8] “Alpha beta crown repository.” [Online]. Available: <https://github.com/Verified-Intelligence/alpha-beta-CROWN>
- [9] “Auto lirpa repository.” [Online]. Available: https://github.com/Verified-Intelligence/auto_LiRPA/
- [10] “Neuralsat: A dpll(t) framework for verifying deep neural networks.” [Online]. Available: <https://github.com/dynaroars/neuralsat>
- [11] “Davis-putnam-logemann-loveland.” [Online]. Available: <https://www.cs.upc.edu/~oliveras/dpll.pdf>
- [12] “Neuralsat repository.” [Online]. Available: <https://github.com/dynaroars/neuralsat>
- [13] “Benchmarks’ archive.” [Online]. Available: https://github.com/ChristopherBrix/vnncomp2023_benchmarks/tree/main
- [14] “Rezultatul complet alpha-beta-crown.” [Online]. Available: https://github.com/IoanaTodoca/VerificareFormalaProiect/blob/main/Versiunea_Finala/rezultat_abcrown.txt
- [15] “Comparatie rezultate alpha-beta-crown.” [Online]. Available: https://github.com/IoanaTodoca/VerificareFormalaProiect/blob/main/Versiunea_Finala/results_ABCROWN.csv

- [16] “Rezultatul complet neuralsat.” [Online]. Available: https://github.com/IoanaTodoca/VerificareFormalaProiect/blob/main/Versiunea_Finala/rezultat_neuralsat.txt
- [17] “Comparatie rezultate neuralsat.” [Online]. Available: https://github.com/IoanaTodoca/VerificareFormalaProiect/blob/main/Versiunea_Finala/results_NeuralSat.csv