Setup

[ ] ↳ 4 cells hidden

Seminar in Deep Neural Network 2025

Paper: It's Not What Machines Can Learn, It's What We Cannot Teach

Gal Yehuda, Moshe Gabel, Assaf Schuster

20th May, 2025

by Sebastian Brunner

Check notebook at go.snb.li/dnn

And GSAT4Bench fork: github.com/Wernerson/G4SATBench

What are the most difficult problems in Computer Science?

NP-hard

NP, NP-hard, NP-complete?

NP (**n**on-deterministic **p**olynomial)

> NP is the set of decision problems *verifiable* in **polynomial** time. (Wikipedia, no date)

Example: Integer Linear Programming

$x_1 + 2x_2 \geq -14$

$-4x_1 - x_2 \leq -33$

$2x_1 + x_2 \leq 20$

Certificate: $x_1 = 10, x_2 = 0$

$10 + 2 \cdot 0 = 10 \geq -14$ ✓

$-4 \cdot 10 - 0 = -40 \leq -33$ ✓

$2 \cdot 10 + 0 = 20 \leq 20$ ✓

NP-hard

> A decision problem H is *NP-hard* when every problem $L \in NP$ can, **in polynomial time**, be reduced to H. (Wikipedia, no date)

$\implies$ H is at least as difficult to solve as all problems in NP.

NP-complete

> NP-hard + NP

- Traveling Salesman
- Longest common subsequence
- Assembling an optimal Bitcoin block (Wikipedia, no date)
- Software Dependency Resolution (Abate *et al.*, 2020)
- Tetris (Breukelaar *et al.*, 2004)

SAT problem

- $(\neg a \vee b) \wedge (a \vee \neg b) \wedge (a \vee b)$ is sat
- $(\neg a \vee b) \wedge (a \vee \neg b) \wedge (a \vee b) \wedge (\neg a \vee \neg b)$ is $unsat$
- $(\neg a \vee b) \wedge (a \vee \neg b) \wedge (a \vee b) \wedge (a \vee \neg b)$ is sat

If we're a really good Super Mario Bros player, we can reduce SAT to SMB! (Aloupis *et al.*, 2012)

How much data for SAT?

$N =$ total number of variables in formula

$L =$ number if literals per clause

$M =$ number of clauses in formula

number of possible clauses $C = \binom{N}{L} \cdot 2^L$

number of possible formulas $F = \binom{C}{M}$

```
display(N_slider, L_slider, M_slider, output)
```

N: 4

L: 3

M: 17

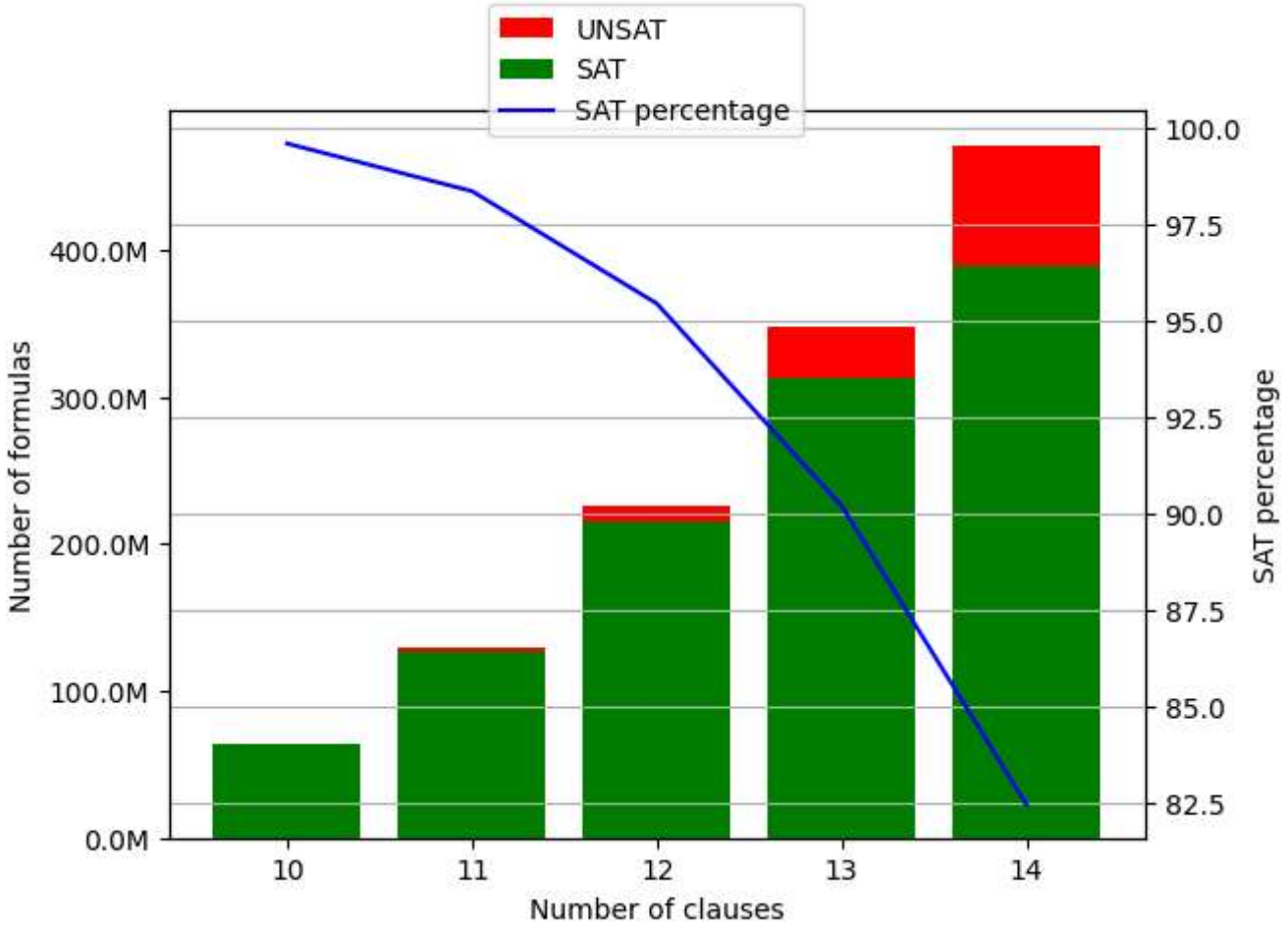
```
Total number of formulas: 566 M (565722720)
This equates to ~81 GB (86555576160 B)
```

Infeasible size & imbalance

Complete 3SAT dataset ($L = 3, N = 4$ fixed)

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import FuncFormatter
def millions(x, pos):
    return '%1.1fM' % (x * 1e-6)


formatter = FuncFormatter(millions)


x = np.array([10, 11, 12, 13, 14])
sat = np.array([64248208, 126906400, 215517608, 313376512, 388824928])
unsat = np.array([264032, 2118080, 10275232, 33997088, 82610672])
pct = np.array([0.9959072572894694, 0.9835838904369155, 0.9544926579602789, 0.9021310542885239, 0.8247678537641197])
```

```
fig, ax = plt.subplots()
ax.bar(x, sat + unsat, color="red", label="UNSAT")
ax.bar(x, sat, color="green", label="SAT")
ax.set_ylabel("Number of formulas")
ax.set_xlabel("Number of clauses")
ax.yaxis.set_major_formatter(formatter)

ax2 = ax.twinx()
ax2.plot(x, pct * 100.0, color="blue", label="SAT percentage")
ax2.set_ylabel("SAT percentage")
ax2.grid()

fig.legend(loc="upper center")
fig.show()
```



M	10	11	12	13	14
formulas	64M	129M	226M	347M	471M
satisfiable	99.6%	98.4%	95.4%	90.2%	82.5%

What now?

Sample & solve!

$\mathbf{SR}(n)$ Sampling (Selsam *et al.*, 2018)

1. Generate clause c_i:
    1. Sample $k \; s.t. \; \mathbb{E}[k] \gtrapprox 4^*$
    2. Sample k variables uniformly w/o replacement $l_{1..k} \sim \mathbf{U}(n)$
    3. Negate each variable with probability 50% $\neg l_i \iff \mathbf{Ber}(0.5) = 1$
2. Add c_i to formula f
3. Check with SAT solver if f is $unsat$
    - if f is sat repeat from 1.
    - if f is $unsat$:
        1. Negate a single variable in c_m to get c'_m
        2. $([c_1, \ldots, c'_m], [c_1, \ldots, c_m])$ is now a $sat, unsat$ pair

$\implies \mathbf{SR}(n)$ produces 50% balanced dataset!

$^*1 + \mathbf{Ber}(0.7) + \mathbf{Geo}(0.4)$

All well that ends well?

Nope!

$\mathbf{SR}(n)$ sampling uses a SAT solver $\implies \mathbf{SR}(n) \notin \mathcal{O}(poly)$

Label Preserving Augmentations

Examples:

- negate all occurences of a variable

- negate all variables
- for sat instances: remove clauses
- for $unsat$ instances: add clauses
- Variable Elimination (Duan *et al.*, 2022)
    - $S_l :=$ set of clauses containing literal l
    - $S_{\neg l} :=$ set of clauses containing literal $\neg l$
    - $S = \{c_1 \otimes c_2 | c_1 \in S_l, c_2 \in S_{\neg l}\}$
    - $(a \vee l) \wedge (b \vee \neg l) \iff (a \vee b)$

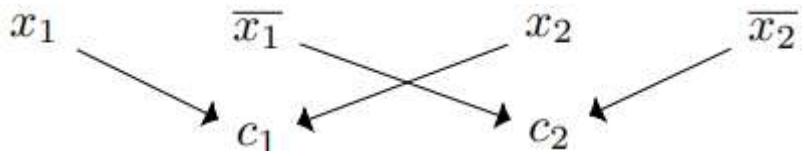
There exist more complex augmentations (Duan *et al.*, 2022).

Model

NeuroSAT as described in 'Learning a SAT Solver from Single-Bit Supervision' (Selsam *et al.*, 2018).

- Single-Bit Supervision $\implies$ only label needed!
- SAT formula represented as graph

Graph for $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$




```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device {device}")

model = Model()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-8)
```

```
Using device cuda
```

Training on augmented $\mathbf{SR}(n \sim U(4, 8))$

Each epoch we sample 500 pairs from $\mathbf{SR}(n \sim U(4, 9))$ (so 1'000 samples).

We augment each sample 99 times to get a total of 100'000 samples per epoch.

Then we train for 150 epochs (~3h on A100 GPU).

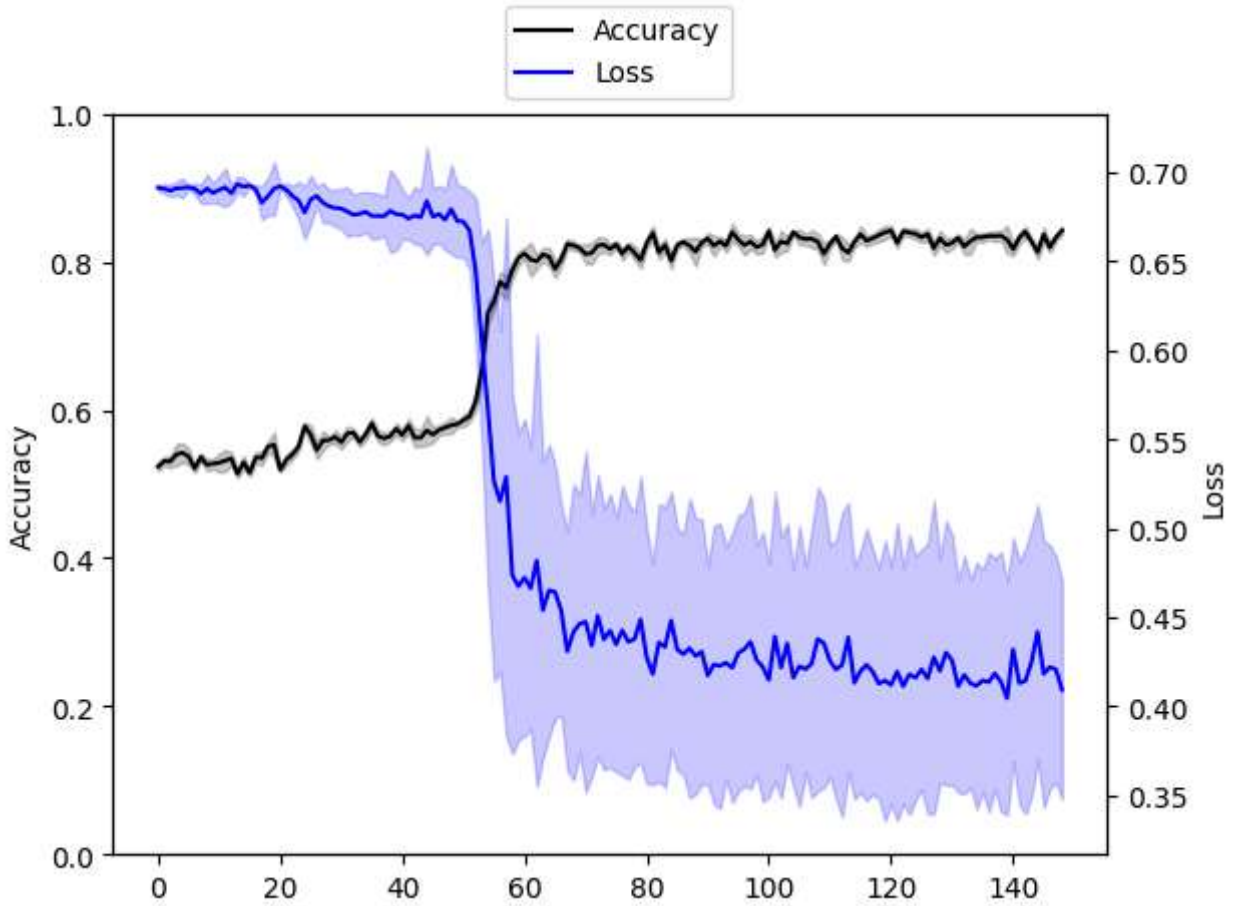
Forked from github.com/zhaoyu-li/G4SATBench.

```
ft = FormatTable()
acc, losses = [], []
for epoch in range(1, 150):
    dataset = augmented_SR(n=U(4, 8), samples=500, augmentations=99)
    train_loader = dataloader(dataset)

    ep_acc, ep_losses = [], []
    model.train()
    ft.reset()
    pbar = tqdm(train_loader, desc=f"Epoch {epoch}")
    for data in pbar:
        optimizer.zero_grad()
        data = data.to(device)
        batch_size = data.num_graphs

        pred = model(data)
        label = data.y
        loss = F.binary_cross_entropy(pred, label)
        ft.update(pred, label)
        pbar.set_postfix(facts(ft))
        ep_acc.append(ft.accuracy())
        ep_losses.append(loss.item())

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
```

```
        optimizer.step()
    torch.save({
        "state_dict": model.state_dict(),
        "epoch": epoch,
        "facts": facts(ft),
        "optimizer": optimizer.state_dict()},
        os.path.join(G_DRIVE_PATH, "model/", f"model_{epoch}.pt")
    )
    acc.append((np.mean(ep_acc), np.std(ep_acc)))
    losses.append((np.mean(ep_losses), np.std(ep_losses)))
    if ft.accuracy() > 0.99:
        break
ft.print_stats()
```

Show hidden output

```
x = np.arange(len(acc))
acc = np.array(acc)
losses = np.array(losses)

fig, ax = plt.subplots()
ax.plot(x, acc[:, 0], color="black", label="Accuracy")
ax.fill_between(x, acc[:, 0] - acc[:, 1], acc[:, 0] + acc[:, 1], alpha=0.2, color="black" )
ax.set_ylim(0., 1.)
ax.set_ylabel("Accuracy")

ax2 = ax.twinx()
ax2.plot(x, losses[:, 0], color="blue", label="Loss")
ax2.fill_between(x, losses[:, 0] - losses[:, 1], losses[:, 0] + losses[:, 1], alpha=0.2, color="blue" )
ax2.set_ylabel("Loss")

fig.legend(loc="upper center")
fig.show()
```



Validation

```
epoch = 149 # epoch you want to load
checkpoint = torch.load(os.path.join(G_DRIVE_PATH, "model/", f"model_{epoch}.pt"), map_location=device)
model.load_state_dict(checkpoint['state_dict'], strict=False)
```

```
<All keys matched successfully>
```

100k independent $\mathbf{SR}(n \sim U(6, 15))$ samples

```
dataset = SR(n=U(6, 15), samples=50_000)
valid_loader = dataloader(dataset, batch_size=512)

model.eval()
ft = FormatTable()
pbar = tqdm(valid_loader, desc="Validating on SR")
for data in pbar:
    data = data.to(device)
    pred = model(data)
    ft.update(pred, data.y)
    pbar.set_postfix(facts(ft))
ft.print_stats()
```

```
Validating on SR: 100%|██████████| 196/196 [03:57<00:00,  1.21s/it, TP=49357, TN=38141, FP=11859, FN=643, TPR=0.987, TNR=0.763,
+--------------------+--------------+-----------------+
|                    |   Labeled 1  |    Labeled 0    |
+--------------------+--------------+-----------------+
| Predicted 1        |         49357|            11859|
| Predicted 0        |           643|            38141|
+--------------------+--------------+-----------------+
| Sensitivity (TPR)  |                        0.987140|
+--------------------+--------------+-----------------+
| Specificity (TNR)  |                        0.762820|
+--------------------+--------------+-----------------+
| Precision (PPV)    |                        0.806276|
+--------------------+--------------+-----------------+
| F-1 Score          |                        0.887588|
+--------------------+--------------+-----------------+
| Overall accuracy   |                        0.874980|
+--------------------+--------------+-----------------+
```

Uniform independent SAT formula $\mathbf{USat}(n, m, l)$

- Variables: $n \sim U(6, 10)$
- Clauses: $m \sim U(20, 30)$
- Literals per clause: $l \sim U(2, 5)$

Unbalanced!

$\gtrapprox 75\%$ satisfiable

```
dataset = USat(n=U(6, 10), m=U(20, 30), l=U(2, 5), samples=100_000)
valid_loader = dataloader(dataset, batch_size = 512)

model.eval()
ft = FormatTable()
pbar = tqdm(valid_loader, desc="Validating on USat")
for data in pbar:
    data = data.to(device)
    pred = model(data)
    ft.update(pred, data.y)
    pbar.set_postfix(facts(ft))
ft.print_stats()
```

```
Validating on USat: 100%|██████████| 196/196 [00:40<00:00,  4.88it/s, TP=68688, TN=20478, FP=3785, FN=7049, TPR=0.907, TNR=0.844
+--------------------+--------------+-----------------+
|                    |   Labeled 1  |    Labeled 0    |
+--------------------+--------------+-----------------+
| Predicted 1        |         68688|             3785|
| Predicted 0        |          7049|            20478|
+--------------------+--------------+-----------------+
| Sensitivity (TPR)  |                        0.906928|
+--------------------+--------------+-----------------+
| Specificity (TNR)  |                        0.844001|
+--------------------+--------------+-----------------+
| Precision (PPV)    |                        0.947774|
+--------------------+--------------+-----------------+
| F-1 Score          |                        0.926901|
```

```
+--------------------+--------------+-----------------+
| Overall accuracy   |                       0.891660|
+--------------------+--------------+-----------------+
```

Complete 3SAT dataset

Dataset of **all possible 64.5 million 3SAT** formulas with 4 variables and 10 clauses.

Highly imbalanced with 99.6% satisfiable formulas.

```
dataset = FileDataset(os.path.join(G_DRIVE_PATH, "validation/3sat_4v_10c/"))
valid_loader = dataloader(dataset, batch_size=2048)

model.eval()
ft = FormatTable()
pbar = tqdm(valid_loader, desc="Validating on 3sat_4v_10c")
for data in pbar:
    data = data.to(device)
    pred = model(data)
    ft.update(pred, data.y)
    pbar.set_postfix(facts(ft))
ft.print_stats()
```

```
Validating on 3sat_4v_10c:  93%|[redacted] | 29206/31501 [2:59:59<13:54,  2.75it/s, TP=5.96e+7, TN=0, FP=233607, FN=16792, TPR=1
```

Balanced subset of 3SAT dataset

```
dataset = FileDataset(os.path.join(G_DRIVE_PATH, "validation/3sat_4v_10c/"), balanced=True)
valid_loader = dataloader(dataset, batch_size=2048)
```

```
model.eval()
ft = FormatTable()
pbar = tqdm(valid_loader, desc="Validating on partial 3sat_4v_10c")
for data in pbar:
    data = data.to(device)
    pred = model(data)
    ft.update(pred, data.y)
    pbar.set_postfix(facts(ft))
ft.print_stats()
```

```
Validating on partial 3sat_4v_10c: 100%|██████████| 258/258 [03:29<00:00,  1.23it/s, TP=263842, TN=0, FP=263967, FN=125, TPR=1,
+--------------------+--------------+-----------------+
|                    |   Labeled 1  |    Labeled 0    |
+--------------------+--------------+-----------------+
| Predicted 1        |       263842 |          263967 |
| Predicted 0        |          125 |               0 |
+--------------------+--------------+-----------------+
| Sensitivity (TPR)  |                        0.999526 |
+--------------------+--------------+-----------------+
| Specificity (TNR)  |                        0.000000 |
+--------------------+--------------+-----------------+
| Precision (PPV)    |                        0.499882 |
+--------------------+--------------+-----------------+
| F-1 Score          |                        0.666456 |
+--------------------+--------------+-----------------+
| Overall accuracy   |                        0.499763 |
+--------------------+--------------+-----------------+
```

	Train	SR	USat	3SAT	Balanced 3SAT
sat%	50%	50%	75%	99.6%	50%
TPR	92%	98%	90%	100%	100%
TNR	76%	76%	84%	0%	0%
Accuracy	84%	87%	89%	99.6%	49.98%

Outcome

So this didn't really work...

And it's not supposed to.

> Given the common assumption that $NP \neq coNP$ we prove that **any polynomial-time sample generator for an NP-hard problem samples, in fact, from an easier sub-problem**. (Yuheda *et al.*, 2020)

$coNP$

> $coNP = \{L | \bar{L} \in NP\}$

$\implies$ every *no*-instance *verifiable* in **polynomial** time

Complete Efficient Samplers Do Not Exist

- **Complete**: the ability to generate every instance with a non-zero probability
- **Efficient**: runs in polynomial time

Incomplete Efficient Samplers are Biased

Proof sketch:

- $SAT \in NP$-hard
- Random sampler S generates SAT problems in $\mathcal{O}(poly)$
- Random sampler S + random seed z = *deterministic* sampler
- given $f \in S$ and z, we can check if f is sat in $\mathcal{O}(poly)$
- $\implies$ classifying if $f \in S$ is sat is in NP

- same for $unsat \implies$ classifying whether $f \in S$ is in $NP \cap coNP$
- $\implies$ classification is strictly easier subproblem because SAT is $\notin NP \cap coNP$

Discussion

Does this mean DNNs cannot learn to approximate NP-hard problems?

It does not. Only that obtaining training and testing data is difficult.

This relates only to (semi-)supervised learning. Reinforcement learning and hybrid approaches (with traditional solvers) are promising.

Is it impossible to use ML to solve hard problems?

No.

Not all problems are NP-hard. Even if, solving easier subproblems can be useful.

The real world is often simpler (or rather more complex) than it is random.

E.g., SAT problems exhibit some kind of structure (Nasser, El Bachir, Mathkour, 2022), which might be exploitable.

Examples:

- Linux Upgradability (Argelich *et al.*, 2010)
- Industrial SAT Instances (Ansótegui *et al.*, 2019)

About the paper...

- Written quite well, easy to follow even though it's quite theoretical
- They have empirical example and a proof, both quite extensive and detailed
- But no code, weights or data :(

- Getting NeuroSAT to run was a pain
- Why we are engineers, not mathematicians
- Great reminder to use meaningful train, test & validation datasets
- Throwing more data at models isn't always the solution

References

Wikipedia (2025) *NP (complexity)*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/NP_(complexity) [Accessed 18 May 2025].

Wikipedia (2025) *NP-hardness*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/NP-hardness [Accessed 18 May 2025].

Wikipedia (2025) *List of NP-complete problems*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/List_of_NP-complete_problems [Accessed 18 May 2025].

Abate, P. *et al.* (2020) 'Dependency Solving Is Still Hard, but We Are Getting Better at It', *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp.547-551

Breukelaar, R. *et al.* (2004) 'Tetris is Hard, Even to Approximate', *International Journal of Computational Geometry and Applications, Volume 14*, pp. 41-68.

Aloupis, G., *et al.* (2015), 'Classic Nintendo games are (computationally) hard', *Theoretical Computer Science, Volume 586*, pp. 135-160

Selsam, D. *et al.* (2019), 'Learning a SAT solver from single-bit supervision', *ICLR 2019*

Duan, H. *et al.* (2022), 'Augment with Care: Contrastive Learning for Combinatorial Problems', *Proceedings of the 39th International Conference on Machine Learning*, pp. 5627-5642

Yuheda, G., Gabel, M. and Schuster, A. (2020), 'It's Not What Machines Can Learn, It's What We Cannot Teach', *Proceedings of the 37 th International Conference on Machine Learning*

Nasser Alyahya, T., El Bachir Menai, M. Mathkour, H., (2022), 'On the Structure of the Boolean Satisfiability Problem: A Survey', *ACM Computing Surveys (CSUR), 55(3)*, Article 46, pp. 1-34

Argelich, J., *et al.*, (2010), 'Solving Linux Upgradeability Problems Using Boolean Optimization', *Electronic Proceedings in Theoretical Computer Science, 29*, pp 11-22

Ansótegui, C., *et al.*, (2019), 'Community Structure in Industrial SAT Instances', *Journal of Artificial Intelligence Research 66*, pp. 443-472
Thanks for listening :)