It’s Not What Machines Can Learn, It’s What We Cannot Teach

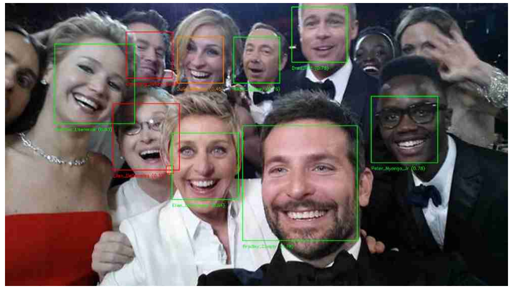
ICML 2020

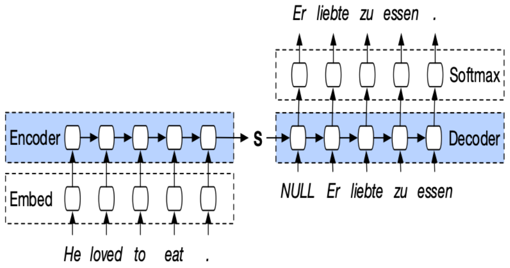
Gal Yehuda, Moshe Gabel, Assaf Schuster

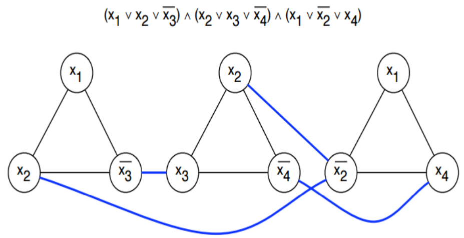
TECHNION Israel Institute of Technology

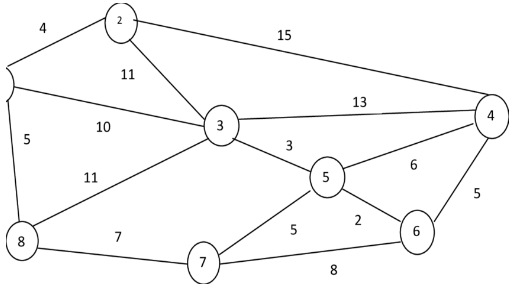
UNIVERSITY OF TORONTO

Applications of machine learning





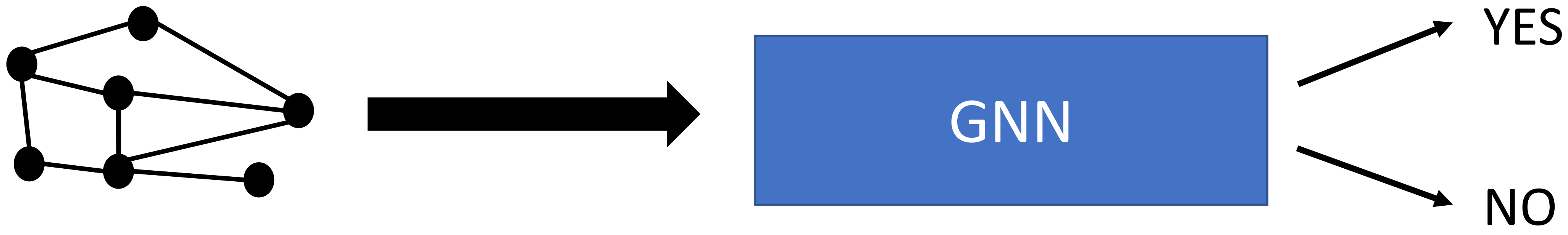




$$\int \frac{x+2}{x^2+2x+5}dx$$

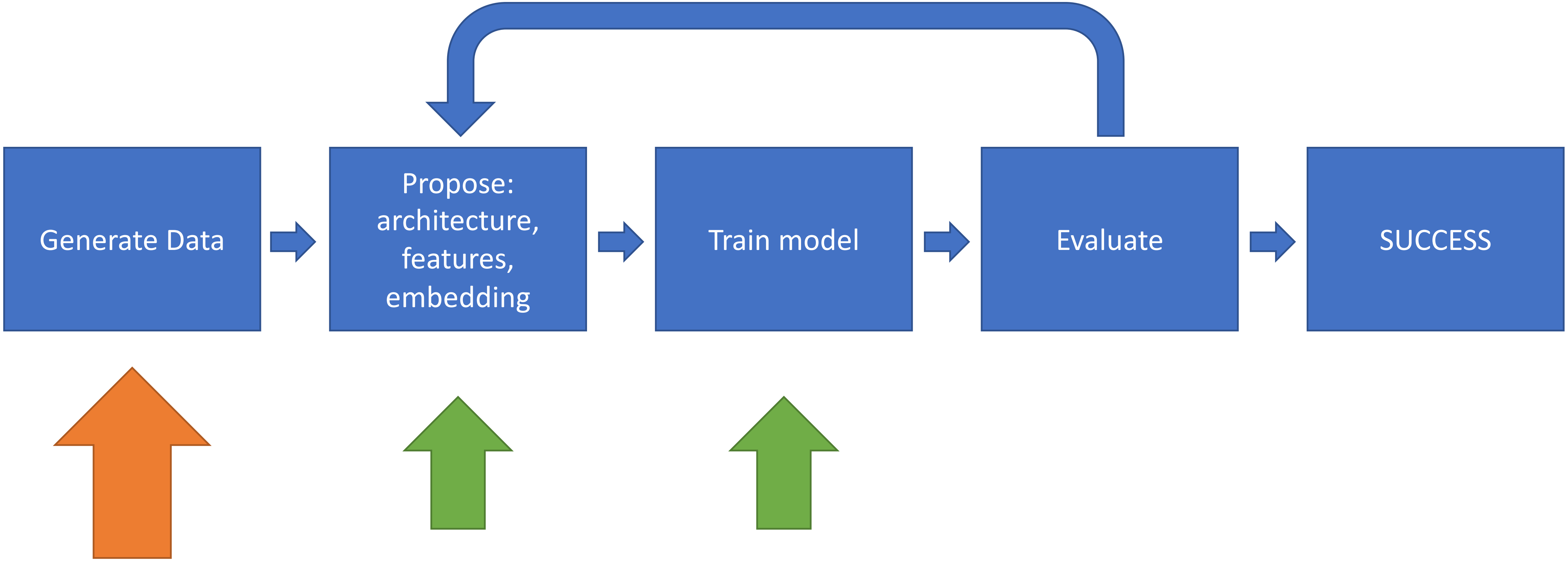
Example : TSP

Given a graph, we feed it to a model which outputs whether a route with cost < C exists



Prates, Avelar, Lemos, Lamb, Vardi,
Learning to Solve NP-Complete Problems - A Graph Neural Network for Decision TSP ,AAAI 2019

The machine learning process



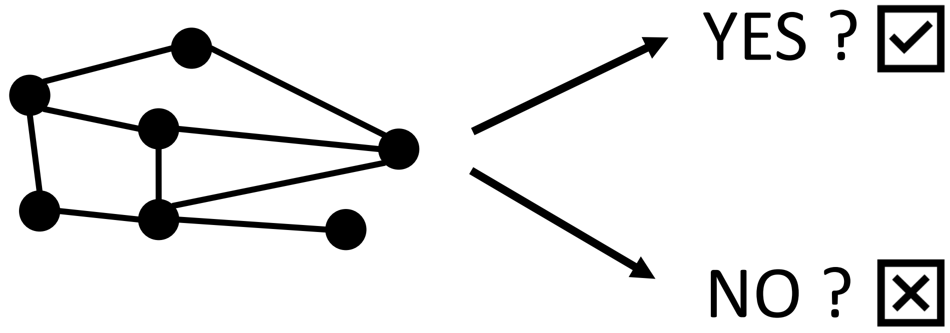
Current Data Generation

SotA ML methods are data hungry

- Need many labeled examples

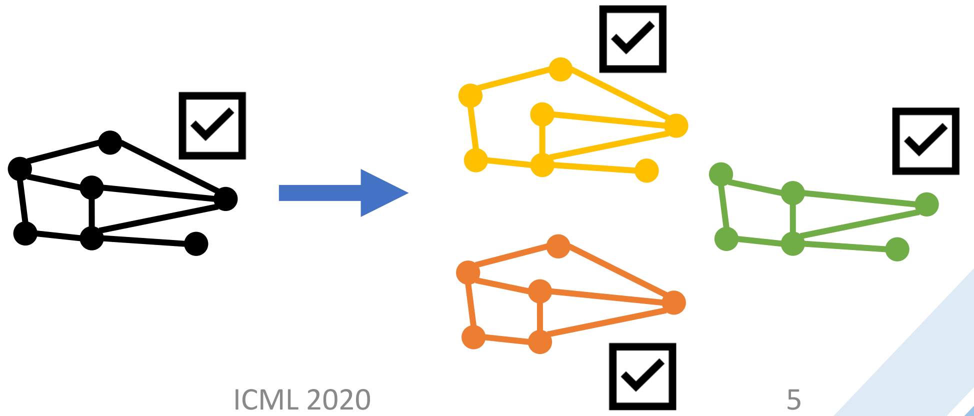
Labeling training data is **slow**

- Need to solve TSP, check 3-SAT, etc.



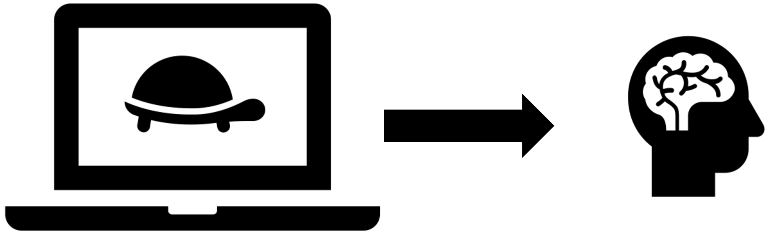
Instead, **data augmentation**:

- Start with small labeled training set
- Apply label-preserving transformation



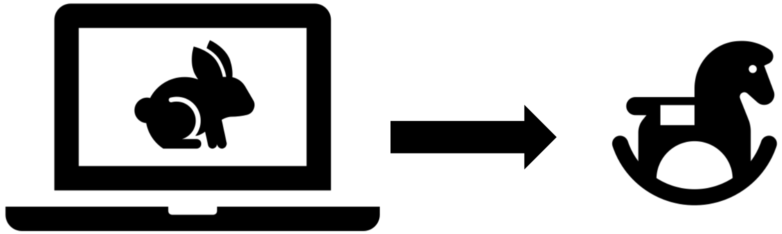
Our Main Result

When starting with NP-hard problem, any efficient data generation or augmentation provably results in **easier subproblem.**



This creates a catch-22:

- Slow data generation → dataset too small
- Fast data generation → easier subproblem



Case Study: Conjunctive Query Containment

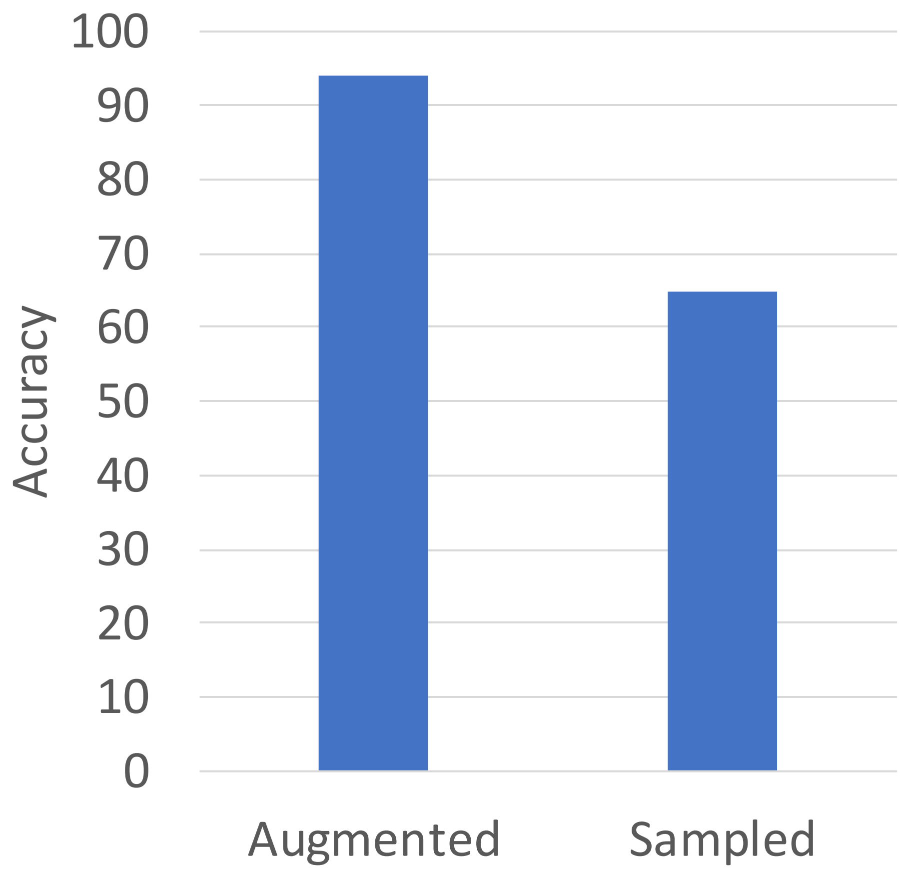
Experiment on a case study, CQC.

Used common data sampling + augmentation approach

Model appears to learn well!

Results on “real” space much lower.

- Up to 30% drop



Takeaways

Efficient data generation results in **easier subproblem** when training.

Can cause **overestimation of accuracy** when testing.

Results in **catch-22**:

- small amounts of training data from right problem?
- or large amounts of training data from easier subproblem?

Let’s dive deeper

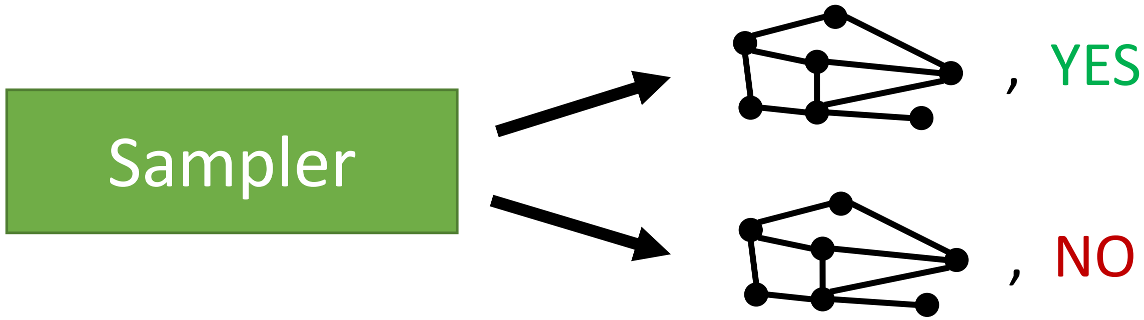
What exactly did we show?

Let **L** be an **NP**-hard language

The binary classification problem: is x ∈ L or not?

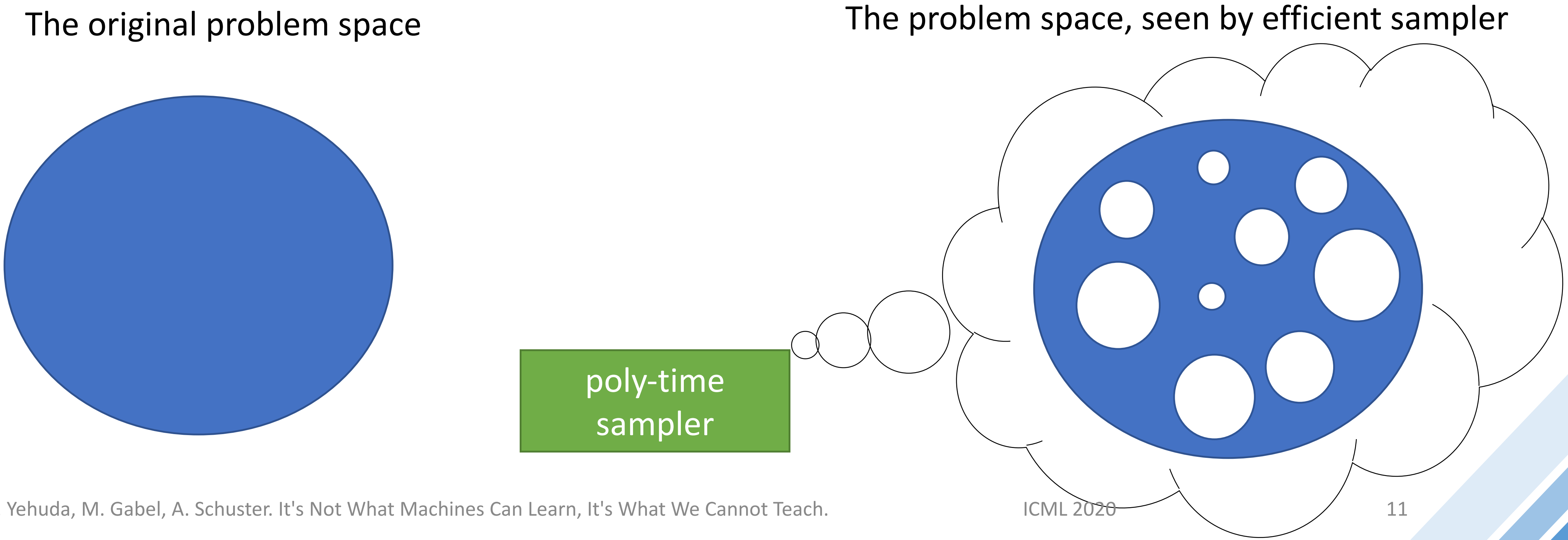
Sampler for L : probabilistic algorithm that generates labeled instances

Efficient Sampler for L : a sampler that runs in poly-time



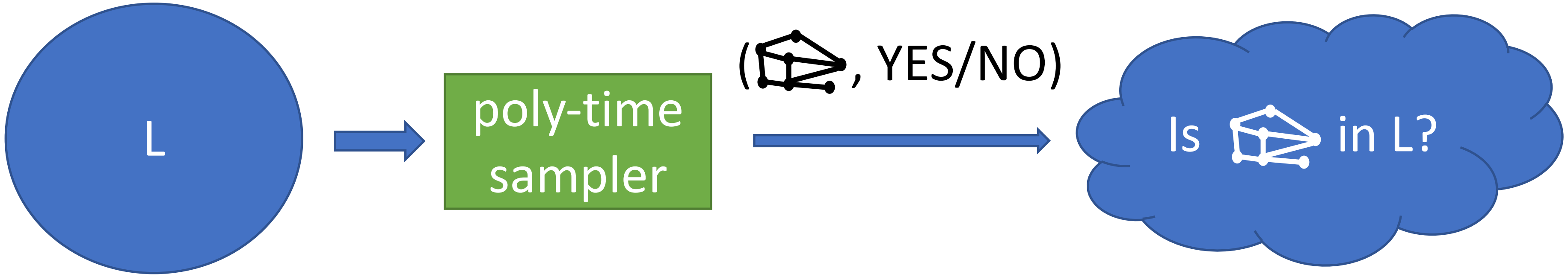
Result 1:
All polynomial time samplers are incomplete

- There are infinitely many instances it cannot generate !



Result 2:
Poly-time sampler yields easier subproblem

If S_L is a polynomial time sampler for a language L, then the classification task over the instances S_L generates is in NP ∩ coNP.



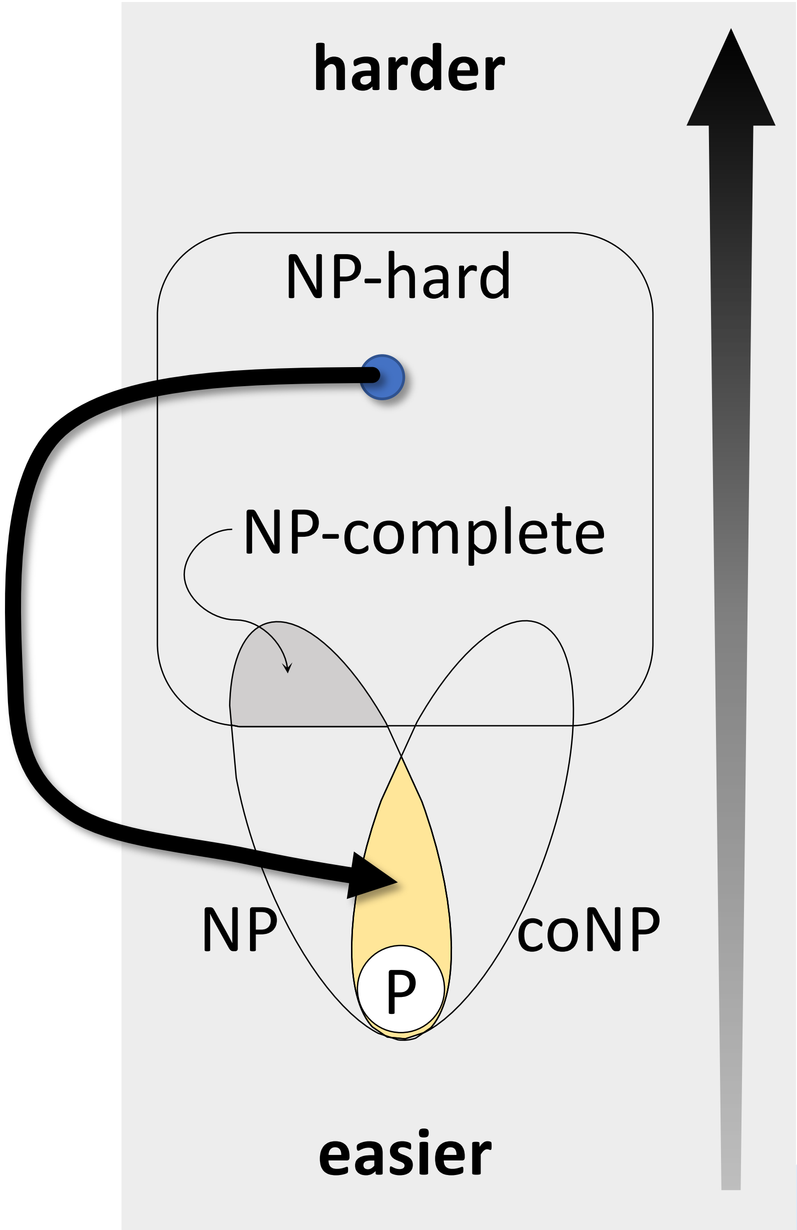
The original problem was NP-hard

Resulting problem is NP ∩ coNP

Meaning: efficient sampling does not preserve hardness

Even if we started with an NP-hard problem,

what's left after an efficient sampling is an easier sub-problem



Proof

NP = easy to verify that $x \in L$

For all x, $\exists u$ such that $M(x,u) = 1 \Leftrightarrow x \in L$

coNP = easy to verify that $x \notin L$

For all x, $\exists u$ such that $M(x,u) = 1 \Leftrightarrow x \notin L$

Proof

If x was generated by an efficient sampler S_L, we can use the randomness used by the sampler both as a membership certificate and a non-membership cetificate

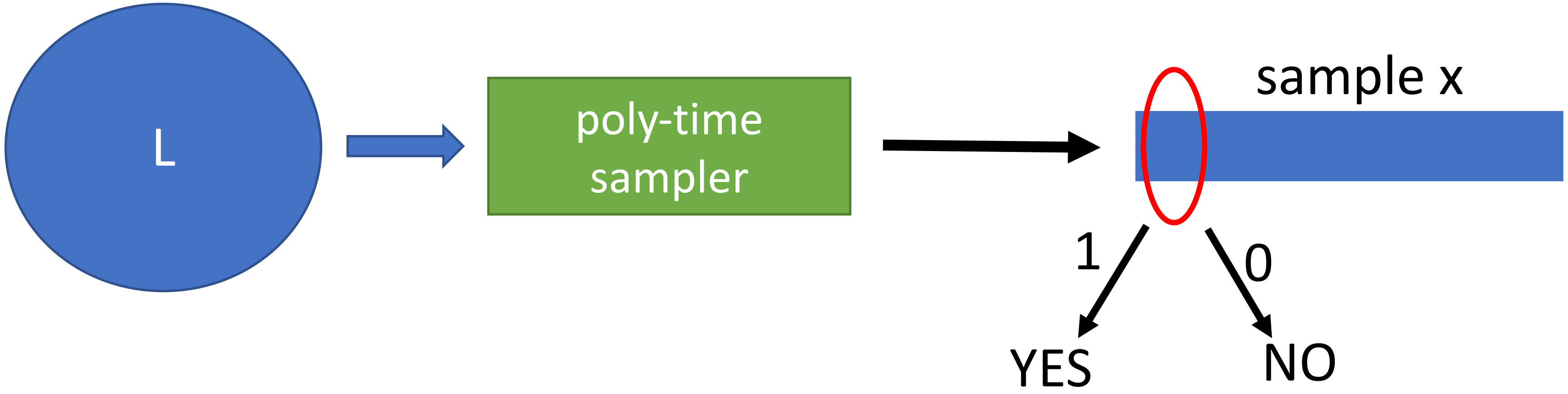
To show that x ∈ L , check if S_L(u) outpus (x, YES) ➔ L ∈ NP

To show that x ∉ L, check if S_L(u) outputs (x, NO) ➔ L ∈ co-NP

Result 3:
It can get really bad...

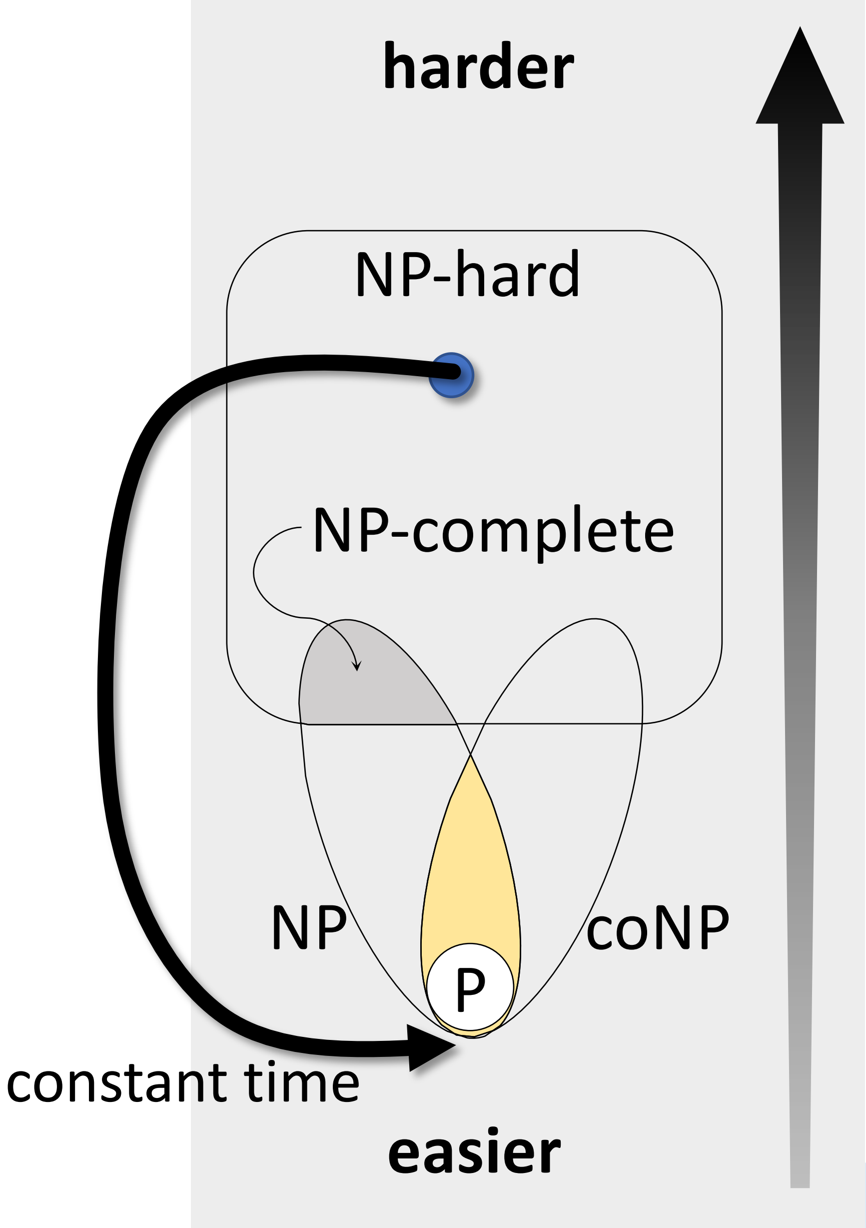
We show an L such that:

1. Original L is NP-hard.
2. Output of **any** polynomial time sampler for L is trivial to classify: the first bit of X is the label with high probability.



It can get really bad...

Meaning: any learning algorithm trained on efficiently generated data ”thinks” it has 100% accuracy, where in fact it learns nothing about the original problem.



Case study: Conjunctive Query Containment

- A conjunctive query **q** over a dataset is a first order predicate of the form:

$$\exists x_1, \ldots, x_n : R_{i_1}(\ell_1, \ell_2, \ell_3) \wedge \cdots \wedge R_{i_s}(\ell_{3s-2}, \ell_{3s-1}, \ell_{3s})$$

- The task: given two queries **q** and **p**, are the results of **q** contained in the results of **p** regardless of database they run on?

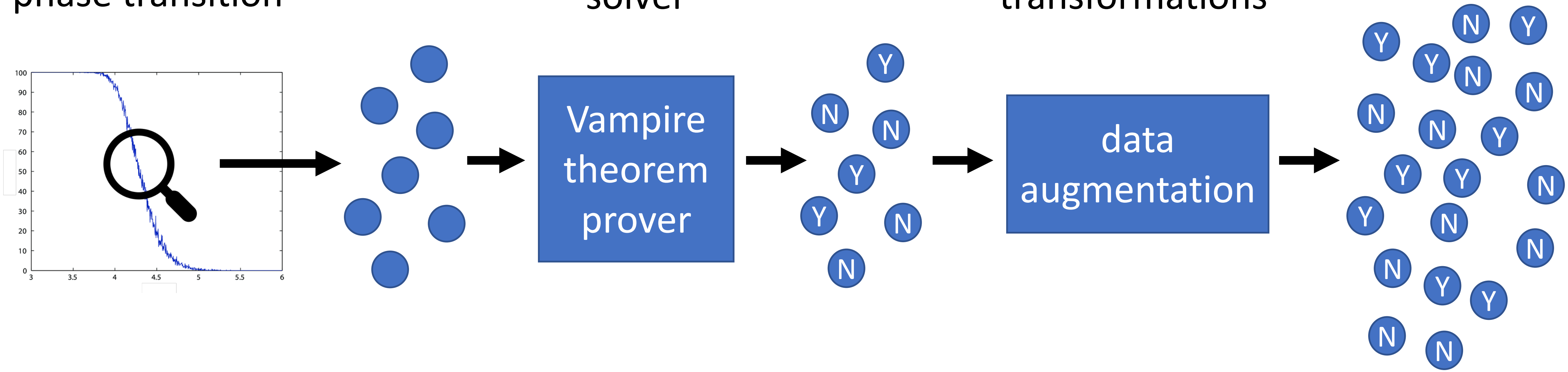
- This is an NP-complete problem.
  - Implications on query optimization, cache management, and more.

Case study: CQC

sample from phase transition

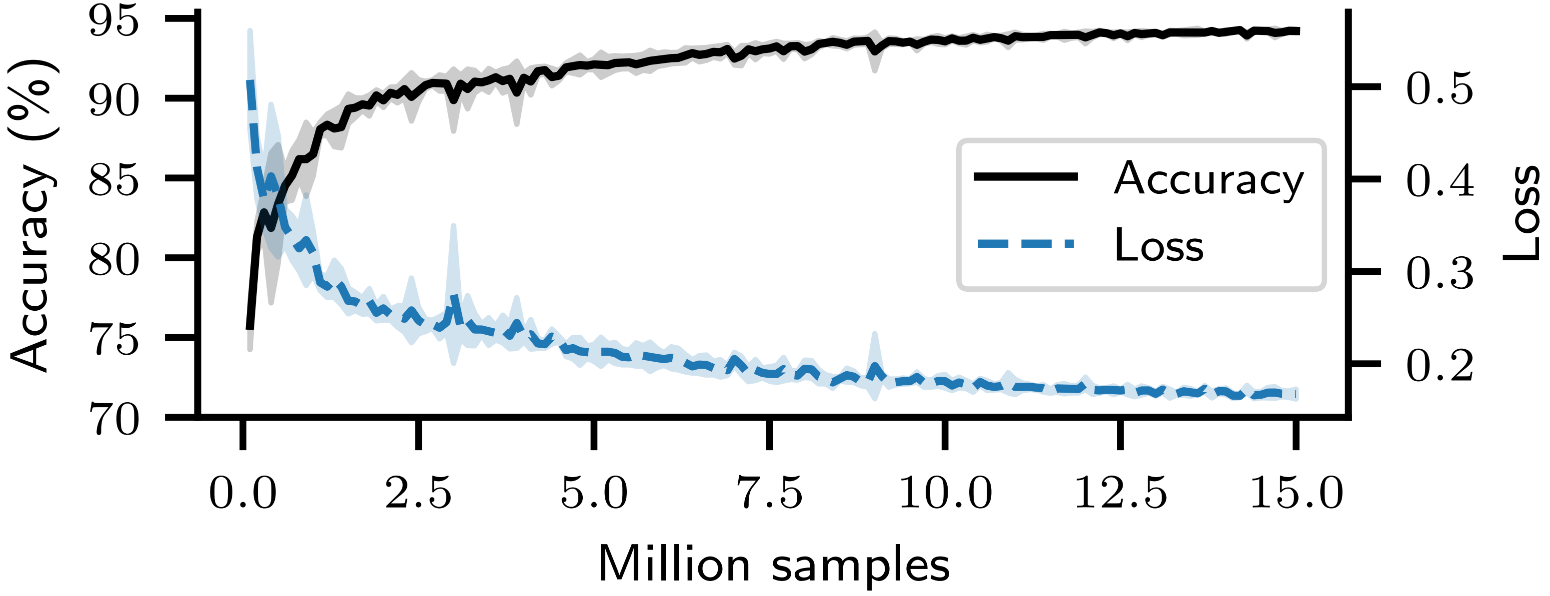
label using solver

label preserving transformations



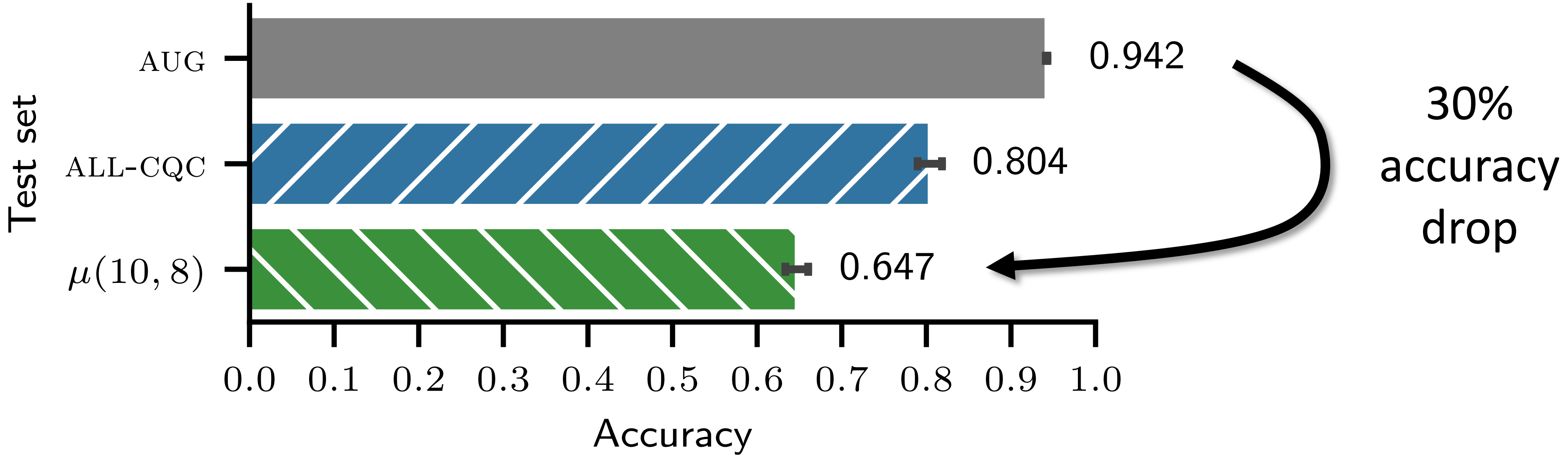
Case study: CQC

Proposed an architecture and trained it to high validation accuracy



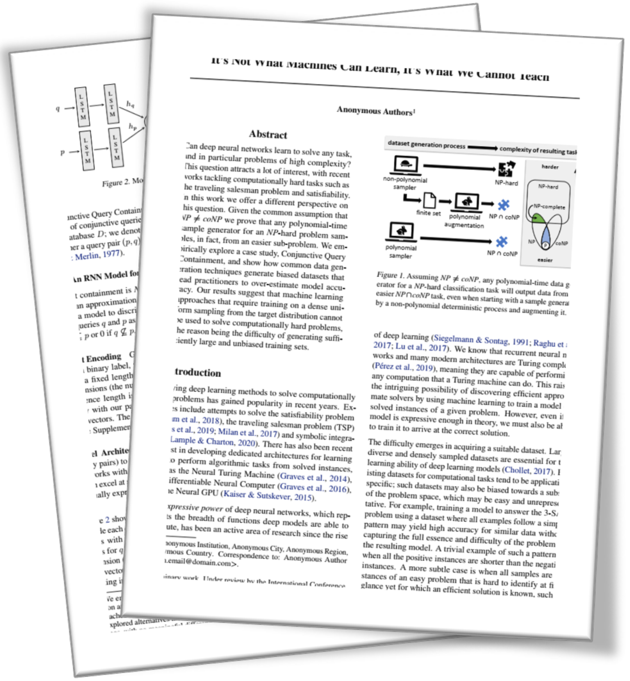
Case study: CQC

Evaluate



In Summary

- Can we use Machine Learning to approximately solve NP-hard problems?
- Not enough to worry about the representation power of the network. Also worry about the procedure used to generate the data.
- **All poly-time data generators result in easier sub-problems.**
    - And it may be very easy.
- We must be careful when we evaluate our models.



THANK YOU!

We will he bappy to discuss the work and answer questions.

ygal@cs.technion.ac.il

mgabel@cs.toronto.edu