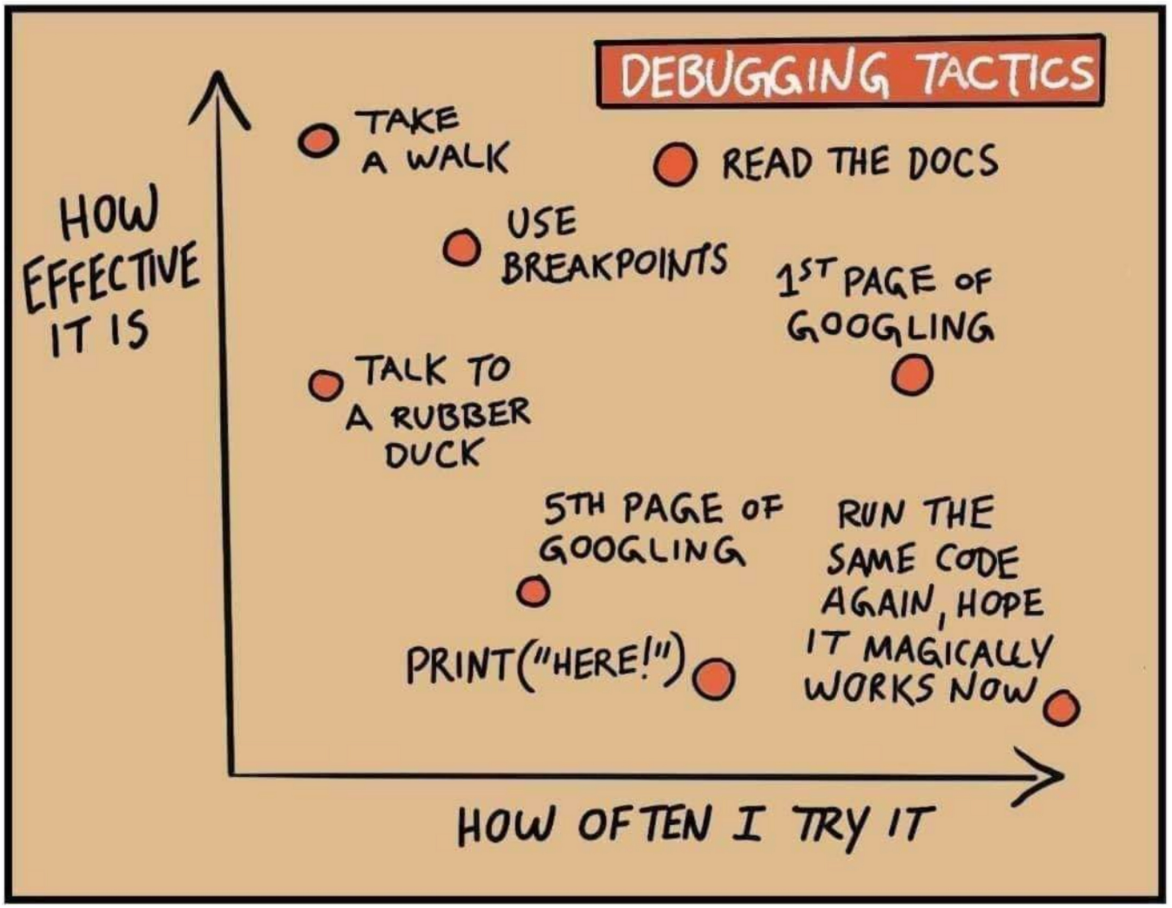
Neurosymbolic Approaches



One-Slide Summary

- Artificial intelligence large language models (e.g., ChatGPT) can propose or edit program artifacts (e.g., source code, invariants, comments). After training they are efficient but may be incorrect.
- Neurosymbolic approaches combine AI and PL substeps into one integrated technique. They are typically more scalable than pure PL techniques and more correct than pure AI techniques.
- We consider LLMs for bug finding, program synthesis (in detail) and program verification (in detail). You can read and understand PL papers.

Terminology: Artificial Intelligence

- **Machine Learning** (ML) is an artificial intelligence subfield of statistical algorithms that can learn from data and generalize to unseen data
- **Natural Language Processing** (NLP) is a machine learning technology that gives computers the ability to interpret, manipulate, and comprehend human language
- **Neural Networks** (NN) are an artificial intelligence method that teach computers to process data in a way that is inspired by the human brain (graphs, weighted edges)

Terminology: Models

- **Sequence Models** transform input sequences (e.g., of words) of one domain (language) into sequences of another domain
- **Generative** AI models create text (not just yes/no) in response to prompts
  - If that text is a program, this is code synthesis
- **Pre-trained** models are trained on (learn from) a large data set of unlabeled text
- Misleading or false results presented as factual by an AI model are **hallucinations**

Terminology: Transformers

- **Transformers** are a neural network sequence model architecture using a notion of "attention" to relate relevant but far-apart tokens in a sequence
  - The wolverine is very fluffy and it is Michigan's mascot.
- **Large language models** (LLMs) are neural network transformer models that can do general-purpose language generation and understanding
- **Chat G P T** is an AI neural network **g**enerative **p**re-trained **t**ransformer large language model

Terminology: Neurosymbolic

- A **symbolic** method approaches such as formal logic, discrete reasoning, or symbolic manipulation. This includes type systems, operational semantics, model checking, abstract interpretation, dataflow analysis, symbolic execution, theorem proving, etc.
- **Neural** methods may specifically refer to methods based on neural networks (e.g., transformers, graph neural networks, recurrent neural networks, etc.) but may also generally refer to any AI method.
- A **neurosymbolic** technique includes both.
  - Why would we want to do that? Let's see!

Motivation #1: “Symbolic” Is Correct But Doesn't Scale

- Software model checking may not terminate, dependent type checking is undecideable in general, abstract interpretation may not converge without mythical widening operators, SAT is NP-Complete, preconditions can't always be computed and VCGen needs annotations for invariants, etc., etc.
- “However, this transformation, grounded in program refinement calculus, is predominantly performed manually, which is time-consuming and error-prone. The necessity of manual code writing makes the program refinement labor-intensive and challenging to automate. Therefore, integrating LLMs’ code generation ability into the refinement process is a logical progression.”
  - *Automated Program Refinement: Guide and Verify Code Large Language Model with Refinement Calculus*

Motivation #2: “Neuro” Scales But Is Incorrect

- “Large Language Models (LLMs) have brought transformative advancements to the fields of language processing and beyond, showcasing exceptional abilities in text generation and comprehension with wide-ranging applications. However, despite their increasing prevalence, LLMs face critical challenges in security and privacy aspects, heavily impacting their effectiveness and reliability. A particularly notable issue among these is the phenomenon of “hallucination”, where LLMs produce coherent but factually inaccurate or irrelevant outputs during tasks like problem-solving. This tendency to generate misleading information not only jeopardizes the safety of LLM applications but also raises serious usability concerns.”
  - *Drowzee: Metamorphic Testing for Fact-Conflicting Hallucination Detection in Large Language Models*

Recent Research

- There are many recent neurosymbolic papers
- We intentionally avoid UM authors today

SEQ2PARSE: Neurosymbolic Parse Error Repair

GEORGIOS SAKKAS, University of California, San Diego, USA
MADELINE ENDRES, University of Michigan, USA
PHILIP J. GUO, University of California, San Diego, USA
WESTLEY WEIMER, University of Michigan, USA
RANJIT JH

We present S
SEQ2PARSE is

Statically Contextualizing Large Language Models with Typed Holes

ANDREW BLINN, University of Michigan, USA
XIANG LI, University of Michigan, USA
JUNE HYUNG KIM, University of Michigan, USA
CYRUS OMAR, University of Michigan, USA

Large language models (LLMs) have reshaped the landscape of program synthesis. However, contemporary LLM-based code completion systems often hallucinate broken code because they lack appropriate code context,

General Research Plan

- Take any problem from Grad PL that was done symbolically and “add an LLM”
  - Direct: just ask ChatGPT to answer it
    - Common in prior years when ChatGPT / Codex / CodeBERT / Cursor / etc. had just come out
  - Modern: help ChatGPT to answer it by adding symbolic information to the LLM input prompt or by using symbolic information to analyze or transform the LLM's output
  - Still Very Rare: train your own LLM model (but maybe with DeepSeek, etc.?)

Example: Neurosymbolic Bug Finding

- Do we always call init before calling use?

Enhancing Static Analysis for Practical Bug Detection: An LLM-Integrated Approach

HAONAN LI,
YU HAO, Univ
YIZHUO ZH
ZHIYUN QIA

- v is a local variable.
- F is the function that declares v.
- U signifies a use operation involving v.
- I represents an initializer that can initialize v.

Then v is *used before initialization* at U, if there exists a potential execution path that makes the use of v, *i.e.*, U ahead of all its initializers I. Note we consider both U and I as function invocations within F. Figure 3 demonstrates

```
1  int F(){
2    int v; // declaration of v
3    if (constraint)
4      init(&v); // initializer of v
5    use(v); // use of v
6  }
```

Fig. 3. A typical example of a UBI bug.

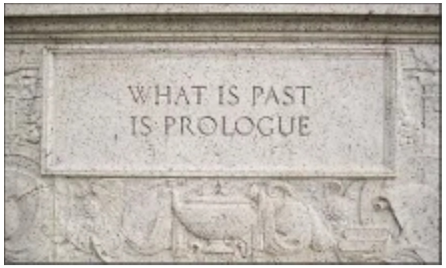
Input Processing →

To tackle challenge C2 (Token Limitation), We employ two strategies: ***(D#2) Progressive Prompt.*** Instead of copying a large number of function bodies (*i.e.*, subroutines), we only provide function details on demand, *i.e.*, when LLMs are not able to conduct a result immediately. ***(D#3) Task***

Output Processing →

majority voting by running each case multiple times and use majority voting to combat stochasticity.

What is Past is Prologue



- Do we always call init before calling use?

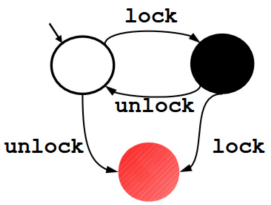
- v is a local variable.
- F is the function that declares v.
- U signifies a use operation involving v.
- I represents an initializer that can initialize v.

Then v is *used before initialization* at U, if there exists a potential execution path that makes the use of v, *i.e.*, U ahead of all its initializers I. Note we consider both U and I as function invocations within F. Figure 3 demonstrates

```
1  int F(){
2    int v; // declaration of v
3    if (constraint)
4      init(&v); // initializer of v
5    use(v); // use of v
6  }
```

Fig. 3. A typical example of a UBI bug.

- Do we always call lock before unlock?



Mon	Model Checking
Jan 13	SLAM Introduction

```
Example ( ) {
1: do{
       lock();
       old = new;
       q = q->next;
2:     if (q != NULL){
3:        q->data = new;
          unlock();
          new ++;
       }
4: } while(new != old);
5:  unlock ();
    return;
}
```

Reading and Understanding PL Papers

- In class activity: we will be taking time during the lecture to read papers, discuss them with our groups, and answer questions about them
- These are available on the course webpage but *some few* relevant pages are included in the guided notes handouts (~1 per pair?)

- Cai et al.'s **Automated Program Refinement: Guide and Verify Code Large Language Model with Refinement Calculus** (POPL 2025)

Question Set #1

Cai et al.'s *Automated Program Refinement*

- Read Pages 1 and 2, then discuss:
- What is the key problem they are solving and why do they claim it is important?
- What are two LLM weaknesses they identify?
- Which two of these best describe the symbolic approach they incorporate?
  - Type Systems - Operational Semantics
  - Axiomatic Semantics - Abstract Interpretation
  - Theorem Proving - Model Checking

Question Set #2

Cai et al.'s *Automated Program Refinement*

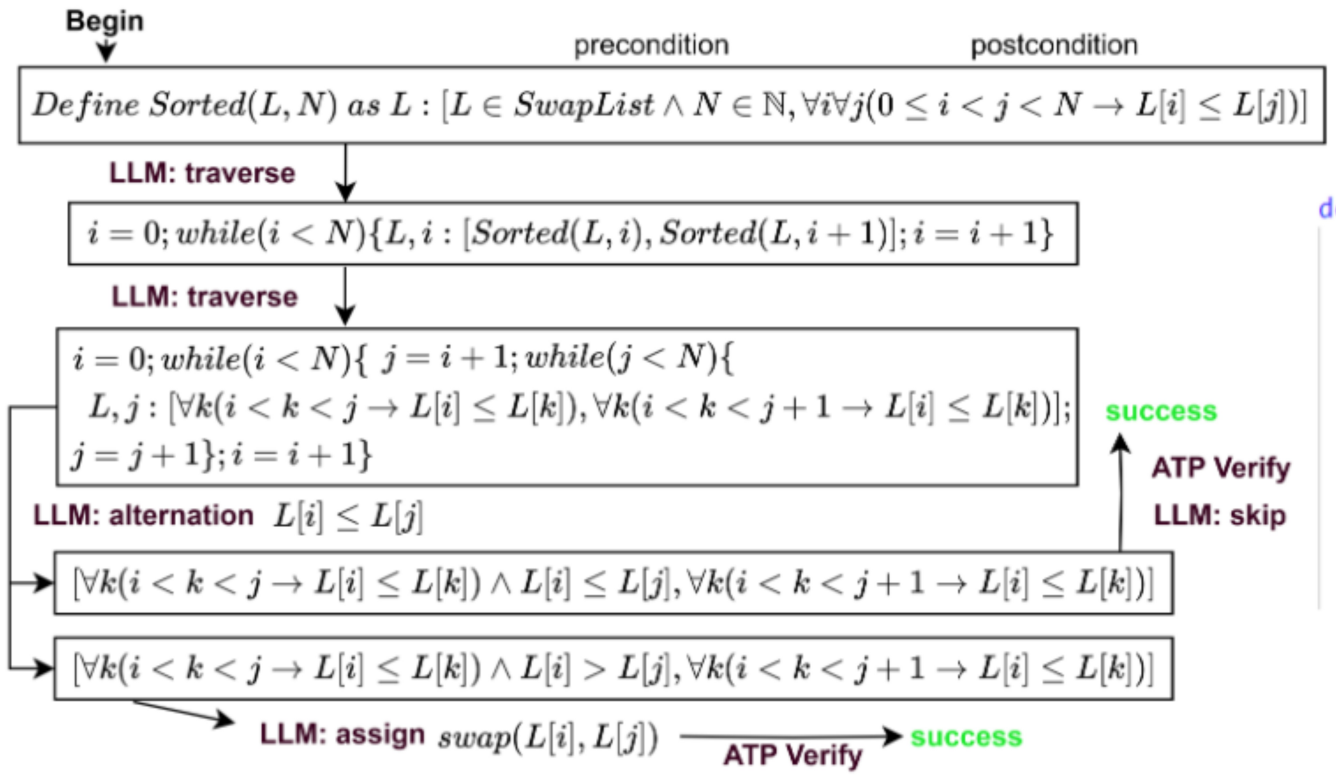
- Read Section 2.1 and Figure 8, then discuss:
- Given that postcondition P is "*return value* >= 0", is A refined by B (A⊑ B), is B refined by A, neither, or both?

```
def A(x):
  return x - 5
```

```
def B(x):
  return x * x
```

- Rewrite the seq-1, alter-1, and assign-1 rules into standard Hoare triple Axiomatic Semantics rules
  - Hint for seq-1 to get you started. Suppose you know precondition x > y and you execute x = y + 1 and you want postcondition x > 5. What must we prove about y for safety?

How Does It Work?




```
def Sorted(L, N):
    i = 0
    while (i < N):
        j = i+1
        while(j < N):
            if L[i] <= L[j]:
                pass
            else:
                swap(L[i], L[j])
            j = j+1
        i = i+1
```

Fig. 10. Bubble sort algorithm with program refinement and the generated pseudo-code. The type of the variables can be extracted from the specification. The generated pseudo code can be translated to a domain-specific language with the formal specification.

Law	LLM	Refine4LLM
Skip	-	verify $P \Rrightarrow Q$
Sequence	M	new spec $[P, M]; [M, Q]$
Assignment	$x = Expr$	verify $P \Rrightarrow Q\langle x := Expr\rangle$
Alternation	G	new spec *if* (G) $(x : [P \wedge G, Q])$ else $(x : [P \wedge \neg G, Q])$
Iteration	I, G	new spec $x : [P, I]$; while(G) do$(x : [I \wedge G, I \wedge (\exists i < M,\ V_i \rightarrow \neg G)])$
Traverse	P	new spec $l : [pre, l[m]]$; i = m; while(i < n) do $(l, i : [P(l,i), P(l,i+1)];\ i = i+1)$

Does It Work?

- They pass 5% more benchmarks than GPT-4 and don't fail any held-out tests (soundness)

Table 6. A comparison of Refine4LLM and popular LLMs on the number of generated programs that passed the test cases in HumanEval and EvalPlus 164 benchmarks. NL means natural language inputs, and FS means formal specification inputs.

Model	LLama3	GPT-3.5	Claude-3	GPT-4		Refine4LLM
Input Specification	NL	NL	NL	NL	NL+ FS	FS
HumanEval Passed	125	126	136	145	148	**150**
EvalPlus Passed	116	116	126	128	142	**150**

- They can synthesize Bubble Sort, Quick Sort, Prime Factorize, Equally Partition, etc.

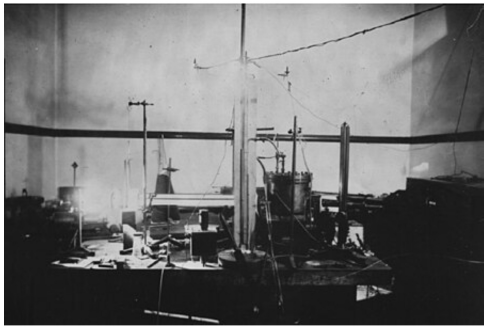
Board Games

- This 2004 board game (Alan R. Moon, Days of Wonder) has sold over 18 million copies in 33 languages. The game has a large-scale travel infrastructure theme and takes place on a graph. Each turn a player either draws more cards, claims a travel route segment, or obtains a new goal route. It has won many awards (including Spiel des Jahres) and is widely praised as an gateway game that is both approachable and strategic.

Cuisine

- This alcoholic beverage is produced by brewing and fermenting starches from malted barley (or another grain). The grain is mashed to convert starch to sugar and dissolved in water to form “wort”. Fermentation by yeast produces ethanol and carbonation. It is one of the oldest and most widely-consumed alcoholic drinks in the world. Most modern versions are brewed with hops, a stabilizing preservative that adds a bitter flavor.

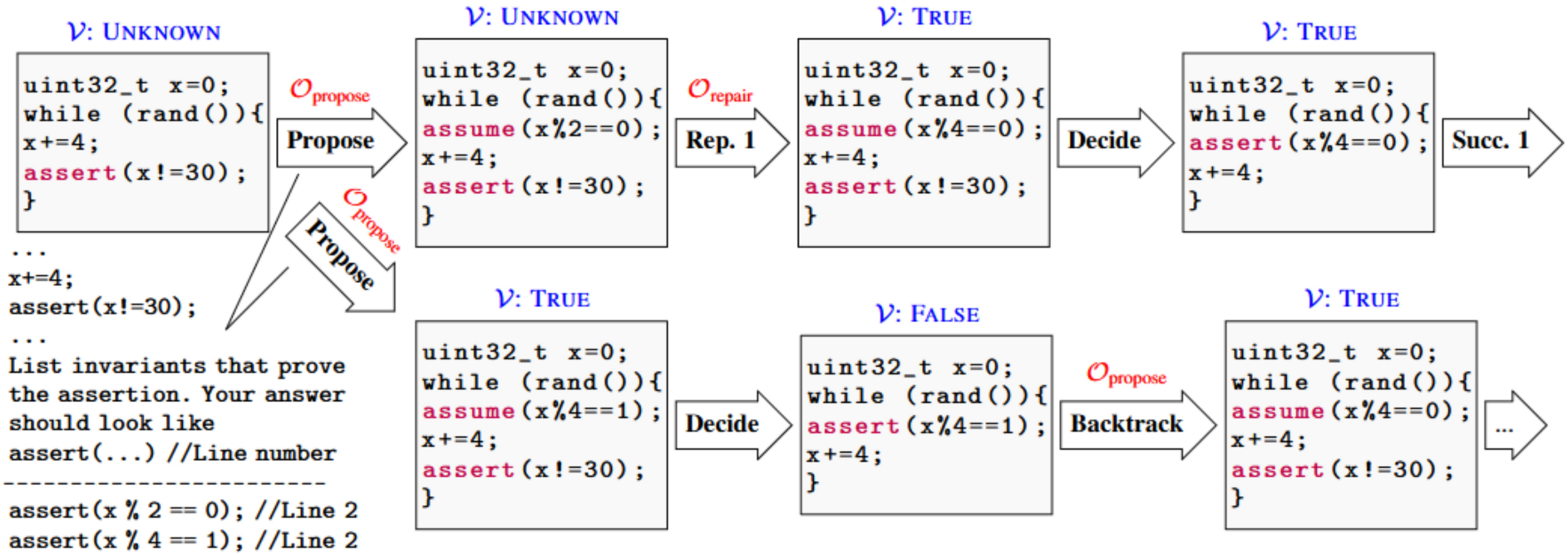
Physics (Nobel Prize)



- In 1909 Millikan and Fletcher performed a famous experiment to measure the charge of an electron. They placed charged *this* between two parallel metal plates, forming a capacitor. A voltage was applied to the plates and adjusted until the *this* was held floating between them. Knowing the mass and gravitational force, it is possible to solve for the electric component: all electric charges were integer multiples of the same value.

Example: Neurosymbolic Formal Verification

- Wu et al.'s **Lemur: Integrating Large Language Models in Automated Program Verification**
  - The LLM proposes and edits candidate invariants



Question Set #3
Wu et al.'s *Lemur: Integrating*

- Read the Introduction and discuss:
- Which of these four properties are used in the Introduction to describe the desired solution and rule out prior work?
  - Automation - Formalized LLM Interaction
  - Efficiency - Integration With Verifier
- Consider: "Specifically, LLMs are employed to propose program invariants in the form of sub-goals, which are then checked by automated reasoners. This transforms the program verification tasks into a series of deductive steps suggested by LLMs and subsequently validated by automated reasoners."
  - Using concepts from the Invariant Detection and Axiomatic Semantics (#2) lectures, explain in prose what happens if the LLM proposes an invariant that is:
  - True and Useful - True but Useless
  - False and Too Strong - False and Too Weak

Question Set #4

Wu et al.'s *Lemur: Integrating*

- Read Section 2 and 3.0 and discuss:
- Does Stable → Invariant? Invariant → Stable?
- When would the verifier return Unknown?
- Is an LLM used for $O_{propose}$, O_{repair}, neither, or both?
- In the Propose rule in Figure 1, what do we know about $\mathcal{V}(\mathcal{P}, \mathcal{A}, q)$? What do we suspect?
  - Note: q is not a typo.
- How do they prove Theorem 3.1 and 3.2?

Does It Work? Quality

- "We found that the LLM-based oracles can produce surprisingly insightful loop invariants that are difficult for conventional formal methods to synthesize. While predicate-abstraction-based techniques typically generate predicates that involve only the operators and values in the program and follow a particular template, LLM is not constrained by these limitations. For example, for the program in Fig. 4, GPT-4 can consistently generate x%4==0 as the loop invariant although the modulo operator is not present in the program."
  - How does this compare to Daikon? DIG? Newton from SLAM?
- "There are also several cases where the LLM generates disjunctive invariants that precisely characterize the behavior of the loops."
  - How does this compare to Daikon? DIG? Newton from SLAM?

Does It Work? Quantity

- They outperform ESBMC (symbolic: bounded model checker), Code2Inv (neural: reinforcement learning), Uautomizer (symbolic: program analysis, prior winner of SV-COMP)

Configurations	Solved	Time	# proposal
Code2Inv	92	–	20
ESBMC	68	0.34	0
LEMUR	107	24.9	4.7

(a) The Code2Inv benchmarks.

Configurations	Solved	Time	# proposals
UAUTOMIZER	0	–	0
ESBMC	0	–	0
LEMUR	26	140.7	9.1

(b) The 50 SV-COMP benchmarks.

- They are also the first approach to include learning that can handle 2+ loops

Time Permitting

- In-Class HW6 Discussion

Homework

- HW6
- Reading for Wednesday