Analyzing and Securing Software via Robust and Generalizable Learning

Kexin Pei

Department of Computer Science

Columbia University in the City of New York

The University of Chicago

“Software is Eating the World”

- Marc Andreessen

Software is Plagued with Errors

"Bad software cost US businesses **$2.41 trillion** in 2022" **- SC Media**

"**280 days** average time companies need to detect and respond to cyber attacks…" **- Skybox**

"Cybercrime is predicted to cost the world **$7 trillion** in 2022" **- CISQ Report**

Hackers breach energy orgs via bugs in discontinued web server

By Sergiu Gatlan November 22, 2022 02:55 PM 0

Florida Hack Exposes Danger to Water Systems

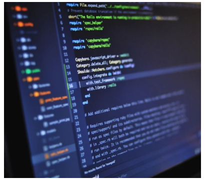
STATELINE ARTICLE | March 10, 2021 | By: Jenni Bergal | Read time: 7 min

CYBERSECURITY

Cyberattack on food supply followed years of warnings

Virtually no mandatory cybersecurity rules govern the millions of food and agriculture businesses that account for about a fifth of the U.S. economy. And now, the risk has become real.

Program Analysis is Crucial for Building Trustworthy Software



Software Programs

Questions

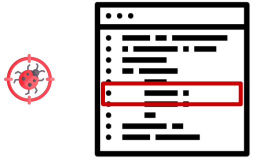
Can pointer p be NULL at line L?

Program Analysis

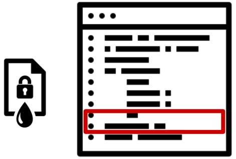
Answers

Help to Build

Vulnerability



Privacy Leak



Trustworthy Software

Security

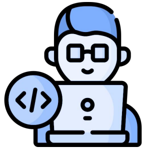
Reliability

Safety

Privacy

Performance

Challenges of Traditional Program Analysis

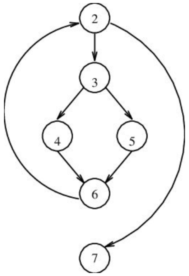


Human Expert

Hand-Curate

Rules and Heuristics

How to



Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

Output: $in(B)$ and $out(B)$ for every basic bloc B.

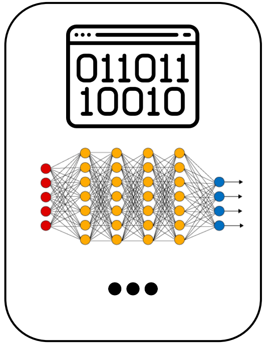
for each B **repeat**
$out(B) := gen(B)$
while changes to any $out(B)$ occur **repeat**
$in(B) := \bigcup_{B' \in \text{pred}(B)} out(B')$
$out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

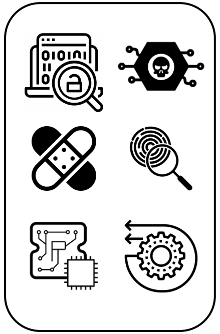
Represent a program?

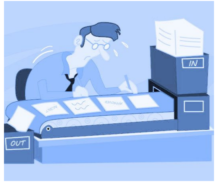
Design analysis rules?

Heterogeneous Software

Various Security Applications







Tune

Significant Manual Effort

Machine Learning Shows Promise for Analyzing Programs

Human Expert

Hand-Curate

Rules and Heuristics

How to

Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

Output: $in(B)$ and $out(B)$ for every basic bloc B.

for each B **repeat**
$out(B) := gen(B)$
while changes to any $out(B)$ occur **repeat**
$in(B) := \bigcup_{B' \in \mathrm{pred}(B)} out(B')$
$out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

Represent a program?

Design analysis rules?

Abundant Public Code

+

Abundant Compute

Learn

Program Representations

Analysis Rules

Machine Learning Shows Promise for Analyzing Programs

CHATGPT

Detecting and Exploiting Vulnerabilities

devtooligan
@devtooligan

omg. seriously mind-blown 🤯

Brandon Rossi
@0xConda

I told ChatGPT to pentest a scope of IPs and it found me $69,420 worth of bugs.

Are there any vulnerabilities in this Solidity contract? I want to make sure no one can steal other peoples' deposits. If there are, how do I fix it?

```
contract Vault {
  mapping(address => uint256) public balances;

  function deposit() public payable {
    balances[msg.sender] += msg.value;
  }

  function withdraw(uint256 amount, address from) public {
    balances[from] -= amount;
    payable(from).transfer(amount);
  }
}
```

This contract appears to have a potential vulnerability in the `withdraw` function. The `withdraw` function allows anyone to specify the `from` address when calling the function, which means that a malicious user could potentially withdraw funds from an arbitrary address by calling `withdraw` and specifying a different `from` address.

To fix this issue, you could modify the `withdraw` function to only allow the caller to withdraw funds from their own account. You could do this by replacing the `from` parameter with `msg.sender`, which will always represent the caller's address. Here is an example of how you could do this:

Machine Learning Shows Promise for Analyzing Programs

Python Profiler

GPT-4

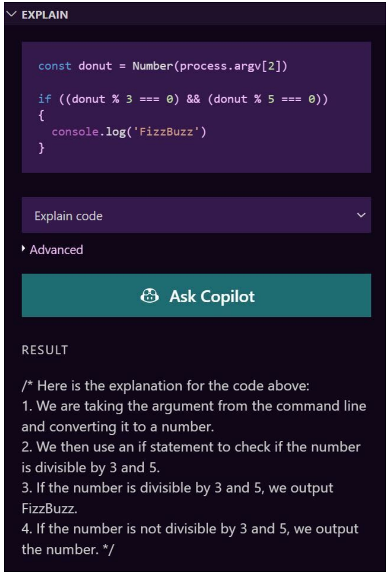
Program Optimization

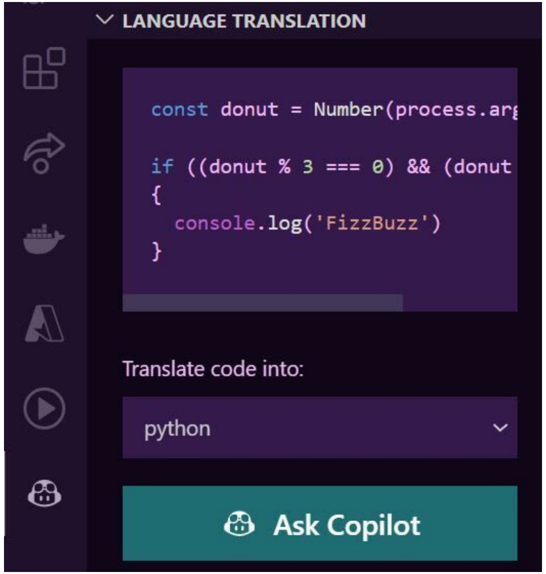
```
z1 = [i for i in range(0,300000)][299999]

# Proposed optimization:
# This code can be optimized by using the built-in function max()
z1 = max(range(0, 300000)) # ~10x faster
```

GitHub Copilot

Explain Code

Translate Code





Google

3% Code Written by ML

ML-Powered Program Fixing, Repair, Refactoring, etc.

Huge Academic Contributions: 500+ Papers
https://ml4code.github.io/

Limitations: Lack Understanding of Program Semantics

A code summarization example (Alon et al., 2019, Yefet et al., 2020, Henkel et al. 2022)
code2vec.org / code2seq.org

```
void f1(int[] array){
  boolean swapped = true;
  for (int i = 0;
    i < array.length && swapped; i++){
    swapped = false;
    for (int j = 0;
    j < array.length-1-i; j++) {
      if (array[j] > array[j+1]) {
        int temp = array[j];
        array[j] = array[j+1];
        array[j+1]= temp;
        swapped = true;
      }
    }
  }
}
```

Prediction: **sort** (98.54%)

```
void f2(int[] ttypes){
  boolean swapped = true;
  for (int i = 0;
    i < ttypes.length && swapped; i++){
    swapped = false;
    for (int j = 0;
    j < ttypes.length-1-i; j++) {
      if (ttypes[j] > ttypes[j+1]) {
        int temp = ttypes[j];
        ttypes[j] = ttypes[j+1];
        ttypes[j+1]= temp;
        swapped = true;
      }
    }
  }
}
```

Prediction: **contains** (99.97%)

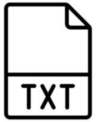
Yefet, Noam, Uri Alon, and Eran Yahav. "Adversarial examples for models of code." OOPSLA 2020.

Common Practice of ML on Code

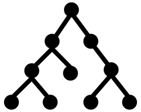
Common Practice:

Static Text

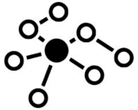
Program



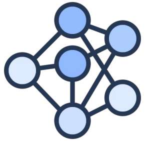
Text Sequence



Trees (AST)



Graphs (CFG/DFG)



ML Model

Specific Task

Program semantics does not just manifest in **static text**

Consequences: Lacking Robustness and Generalization

I. **Overfit** to **spurious textual** and **task-specific** patterns

II. **Distribution shift**: **program syntax** and **task requirement** changes

Security Applications Require More Rigorous Understanding of Program Semantics

A Popular Type of Program Analysis in Security: Binary Analysis

Program Analysis

Binary Analysis

Stripped Binaries

Proprietary software,
Third-party component
Malware

Additional Complications:

```
int variable_init()
{
    int var = 22;
    return var;
}
```

```
mov eax,0x16
```

```
xor eax,eax
sub eax,-0x16
```

Variables
Arguments
Types
Data Structures
Names
...

Semantic Hints are Absent

Registers
Memory Accesses

Various Code Transformations

GCC Visual C++

Compilers

Operating Systems

x64 x86

Architectures

-O0 -O1
-O2
-O3 -Od Ox

Optimizations

Obfuscations

What do Robustness and Generalization Imply in Binary Program Analysis?

Same Source Code

GCC Visual C++

Compilers

Operating Systems

x64 x86

Architectures

-O0 -O1 -O2
-O3 -Od Ox

Optimizations

Obfuscations

Code Transformations that Alters Only Syntax

Same High-Level Semantics

Robustness: Stay Invariant to **Syntactic** Changes

Generalization: Generalize to new **Syntactic** Changes

Security Applications Require Rigorous Understanding of Program Semantics

Detecting Binary Code Reuse Vulnerability

Vulnerable Code → **Detect Similar Code** →

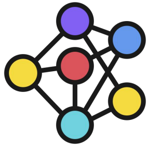
Binary-only firmware with third-party components

```
mov eax,0
add eax,0x16
```

eax=0
eax=0x16

```
xor eax,eax
sub eax,-0x16
```

eax=0
eax=0x16



ML Model → **Not Similar**

KP

```
mov eax,0
add eax,0x16
```

This is assembly code…these instructions **initialize the EAX register to 22**.

KP

is it similar to

```
xor eax,eax
sub eax,-0x16?
```

No...the second set uses the "`xor`" and "`sub`" instructions to **set the value of EAX register to -22**...

Without understanding **mov**, **xor**, **add**, **sub**, etc.

ML model cannot reason about program behavior to predict similarity

My Research

Data-Driven Program Analysis

Program Behavior

Semantic Similarity
[TSE'22]

Specification Inference
[ICML'23]

Debug Symbol Recovery
[FSE'21, CCS'22]

Memory Dependence
[FSE'22]

Efficient 98.1×

Precise 118%+

Generalizable and Robust Across

GCC, Visual C++ — Compilers

x64, x86 — Architectures

-O0 -O1 -O2 -O3 -Od Ox — Optimizations

Obfuscations

TALOS, Object Security (Secure technology), Google, GHIDRA

Program Structure

Type Inference
[FSE'16]

Fuzzing via Program Smoothing
[Oakland S&P'19]

Malware Analysis
[DSN'15]

Attack Forensics
[ACSAC'16]

Disassembly
[NDSS'21]

SSL/TLS Hostname Verification
[Oakland S&P'17]

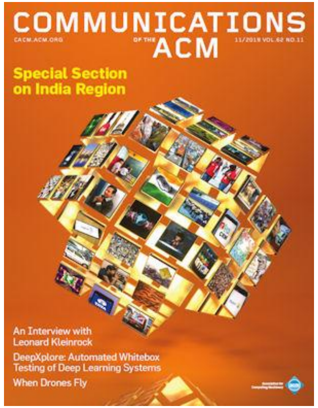
My Research

Data-Driven Program Analysis

Systematic Whitebox Testing of Neural Networks
[SOSP'17, ICSE'18]

SOSP Best Paper Award

Inspired over Thousands of Follow-Up Projects



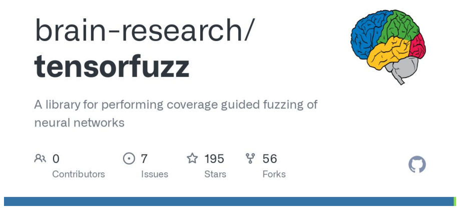
techradar.

IEEE SPECTRUM

SCIENTIFIC AMERICAN

Newsweek

CSAW'18
APPLIED RESEARCH COMPETITION



Formal Verification of Security Properties of Neural Networks
[Usenix'18, Neurips'18, DeepTest'18]

My Research

Data-Driven Program Analysis

Semantic Similarity
[TSE’22]

Specification Inference
[Current]

Debug Symbol Recovery
[FSE’21, CCS’22]

Memory Dependence
[FSE’22]

Learning Execution Semantics for (Binary) Program Analysis

Type Inference
[FSE’16]

Fuzzing via Program Smoothing
[NDSS’21]

Malware Analysis
[DSN’15]

Attack Forensics
[ACSAC’16]

Disassembly
[NDSS’21]

Systematic Whitebox Testing of Neural Networks
[SOSP’17, ICSE’18]

Formal Verification of Security Properties of Neural Networks
[Usenix’18, Neurips’18, DeepTest’18]

Learning Execution Semantics for Binary Program Analysis

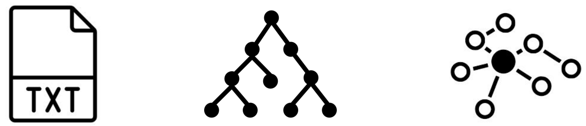
Security Applications Require Rigorous Understanding of Program Semantics

Common Practice:

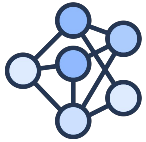
```
mov eax,0
add eax,0x16
```

```
xor eax,eax
sub eax,-0x16
```

Static Text



Text Sequence Trees (AST) Graphs (CFG/DFG)



ML Model

Similarity Detection

Program semantics does not manifest in **static text**

Banana is a **fruit**
Orange is a **fruit**

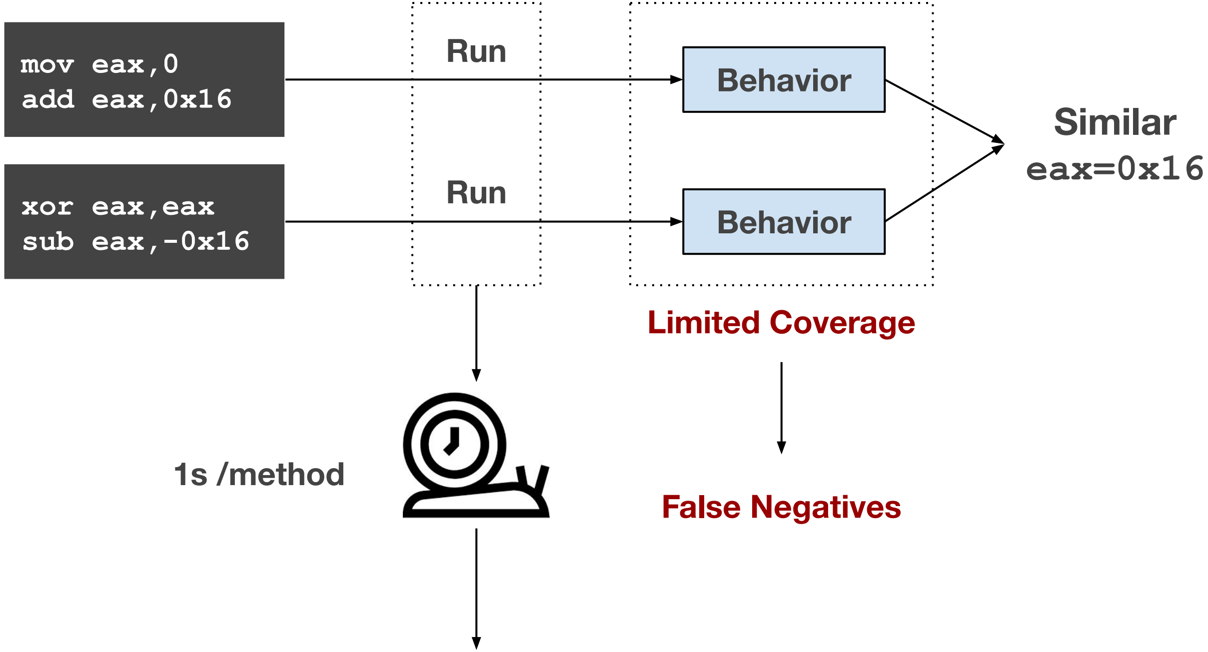
Semantics(banana)=
Semantics(orange)=
Fruit

```
mov eax, 0
add eax, 0x16
```

Semantics(mov)=
Semantics(add)=
Unknown

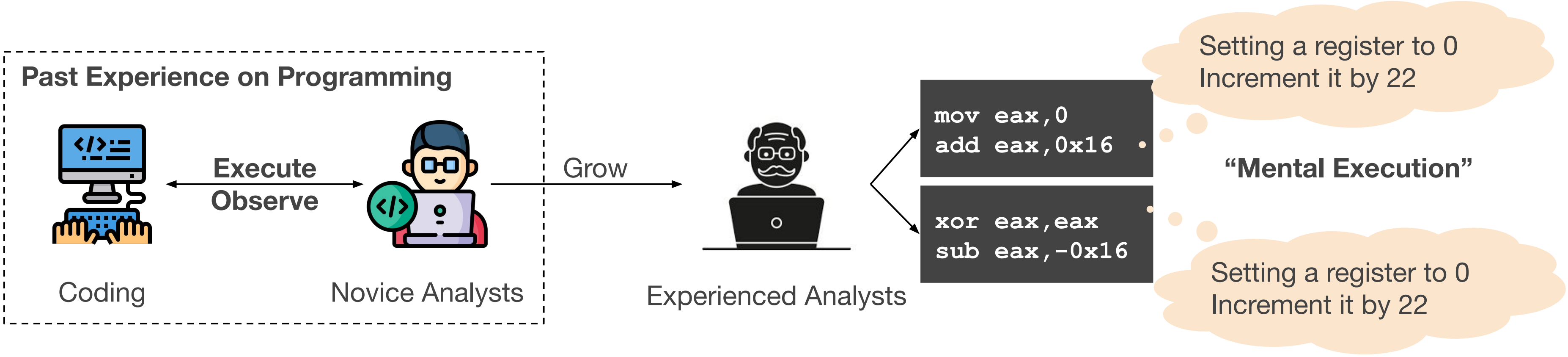
Lack understanding of instruction semantics, so cannot reason about the program behavior

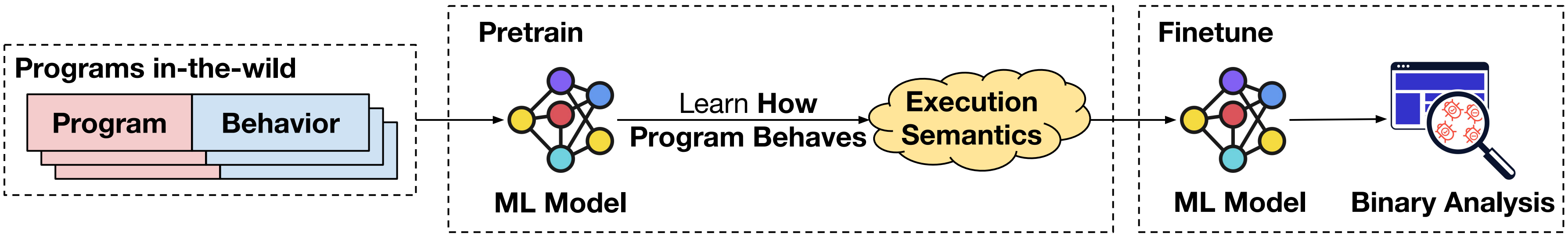
Why not dynamic analysis?



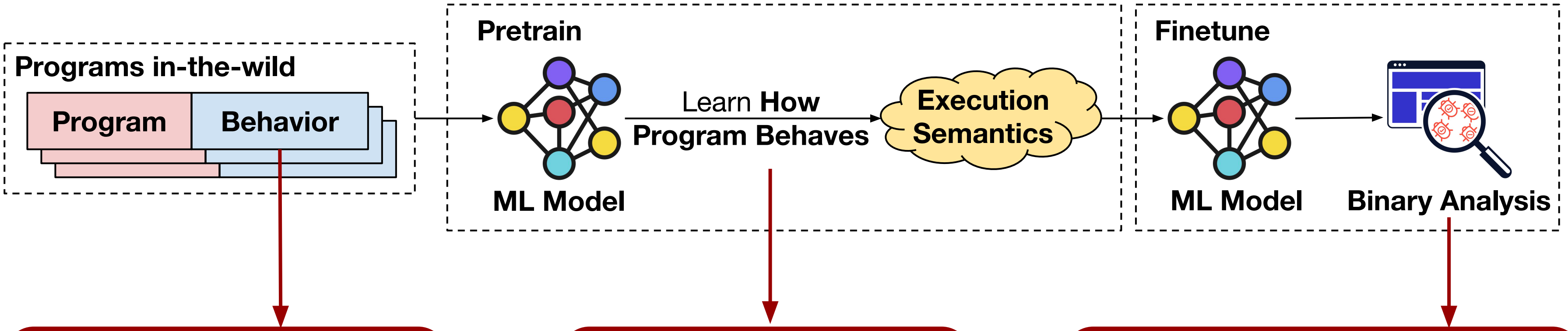
Querying **1M+** Firmware Functions Takes **11+ Days**

Learning Execution Semantics and Transferring it without Dynamic Analysis





Challenges of Learning Execution Semantics

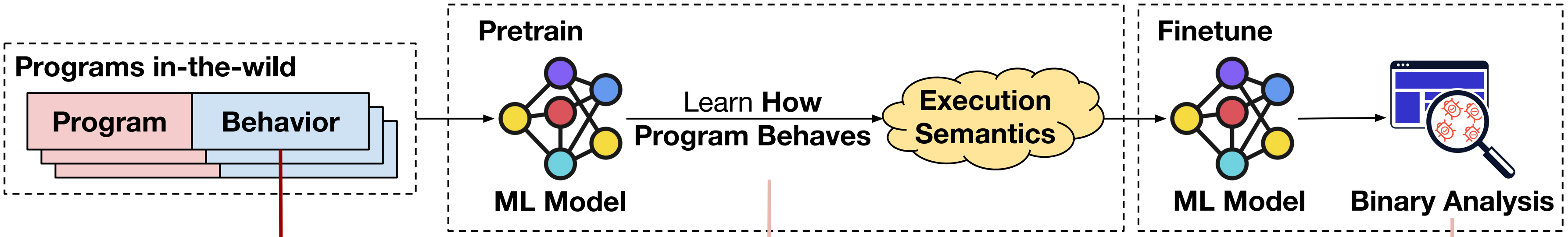


1. How to collect and represent the diverse program behavior?

2. How to train the model to reason about program behavior?

3. How to avoid the expensive dynamic analysis?

Challenges of Learning Execution Semantics

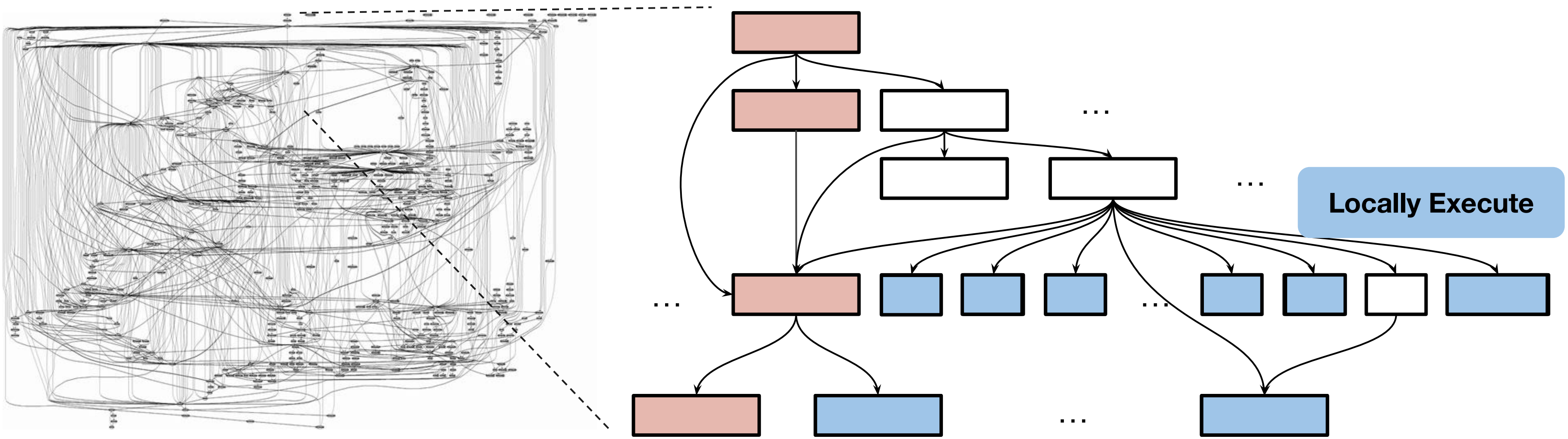


1. How to collect and represent the diverse program behavior?

2. How to train the model to reason about program behavior?

3. How to avoid the expensive dynamic analysis?

How to Collect Diverse Program Behaviors?



Microsoft IIS Call Graph

Under-Constrained Micro-Execution:
Specify arbitrary code piece to execute

- **Expose diverse code behaviors**
- **Benefit large-scale pretraining on diverse execution behavior**

How to Collect Diverse Program Behaviors?

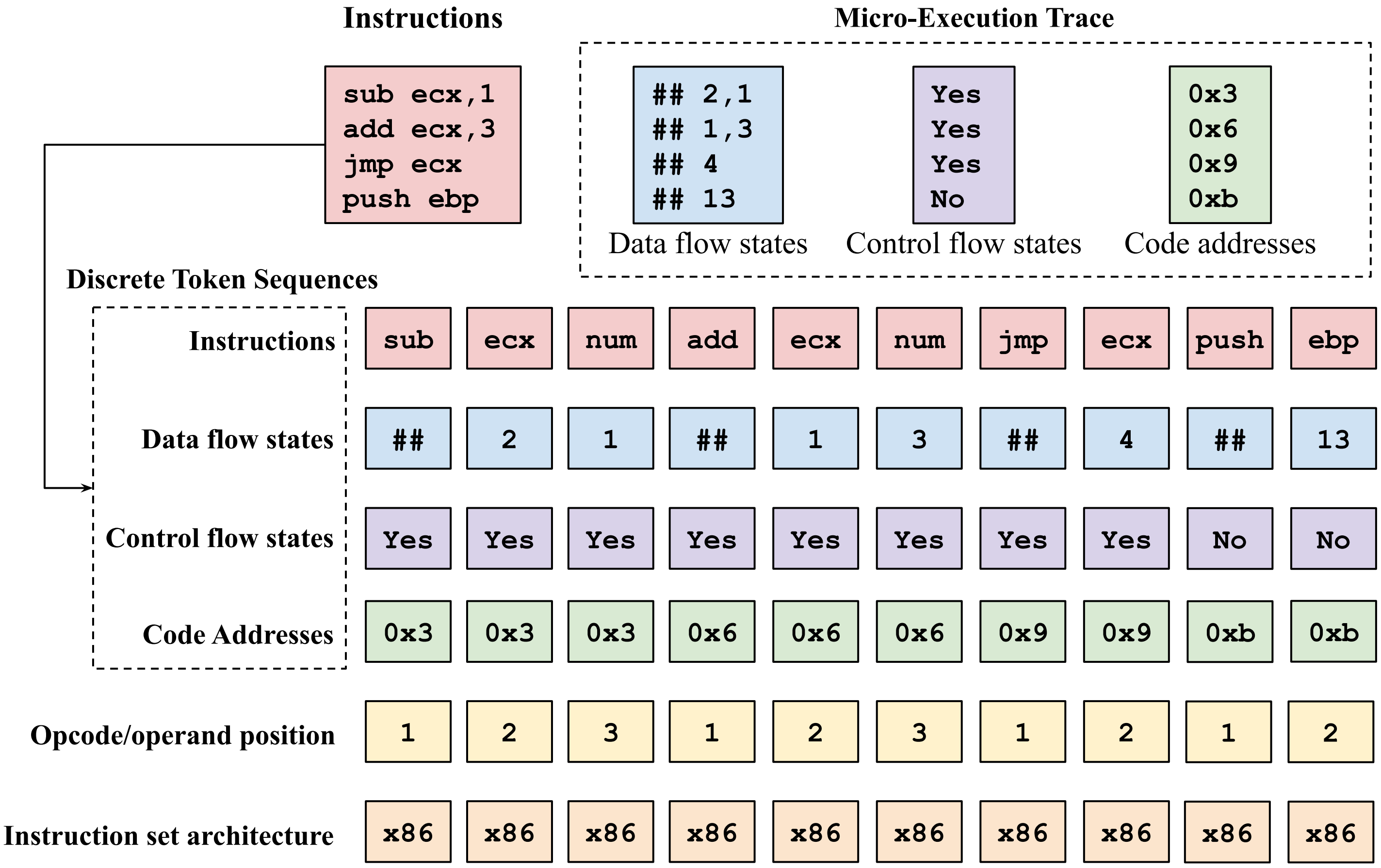
Program instructions

```
......
0x1c: mov ebp,esp
0x1f: add [ebp+0x8],0x3
0x26: cmp [ebp+0x8],0x2
0x2d: jle 0x3a
0x33: add [ebp+0x8],0x1
0x3a: mov eax,0x1a
......
```

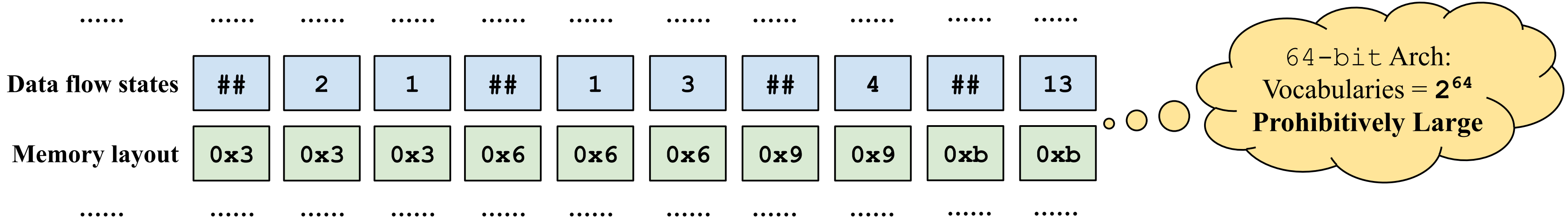
Data flow states	Control flow states	Code Addresses
......	......	......
## 0xc,0x4	✓ Yes	0x1c
## [0xc+0x8],0x3	✓ Yes	0x1f
## [0xc+0x8],0x2	✓ Yes	0x26
## 0x3a	✓ Yes	0x2d
## [0xc+0x8],0x1	✗ No	0x33
## 0x1,0x1a	✗ No	0x3a
......	......	......

Aligned with Program Instructions

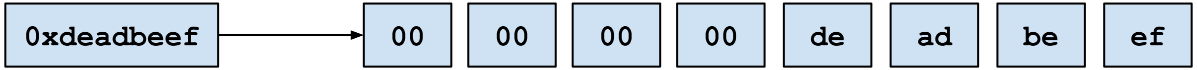
Input Representation



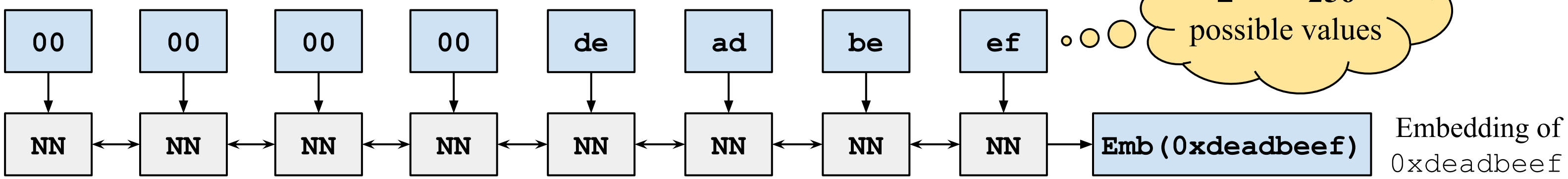
Numerical Representation



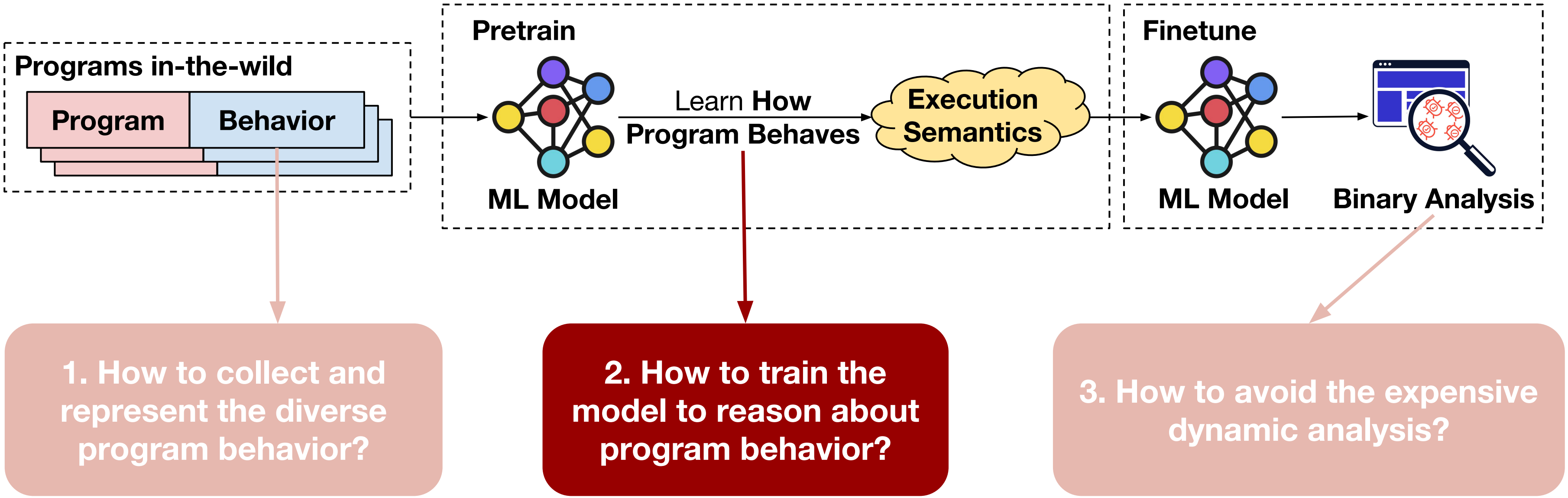
Pad Each Numeric Token as a Fixed-Length 8-Byte Sequence:



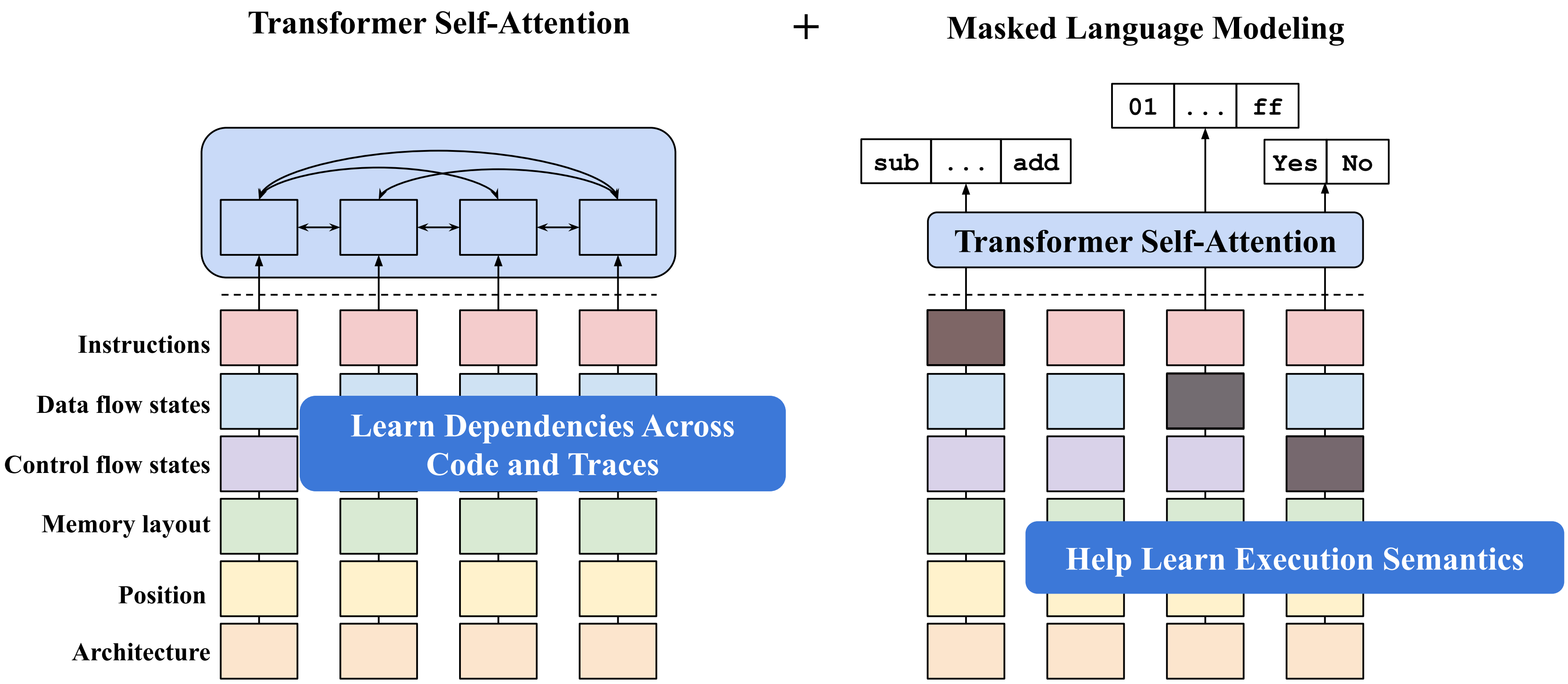
Learning to Represent the Value with a Neural Network:



Challenges of Learning Execution Semantics

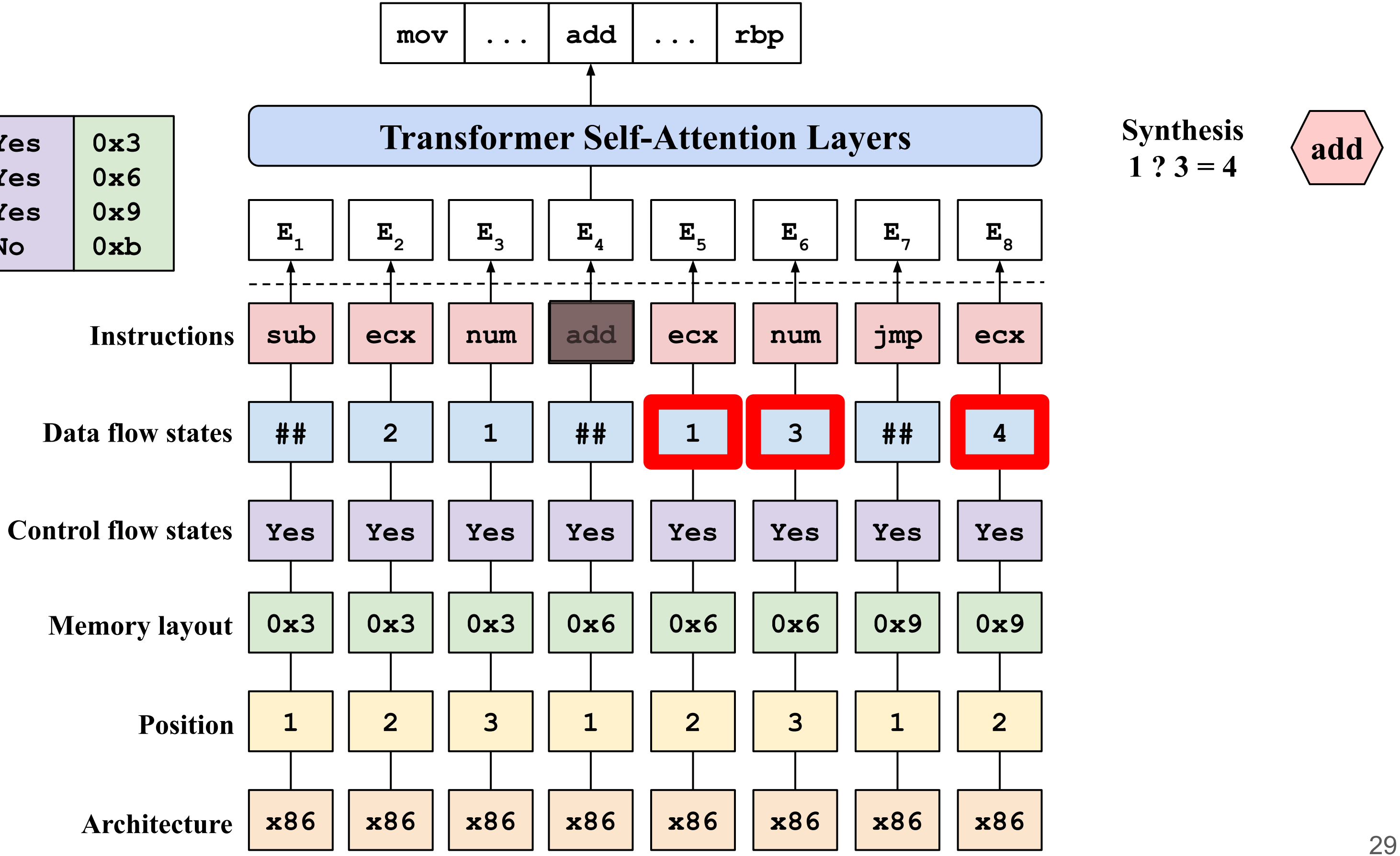


How to train the model to reason about program behavior?



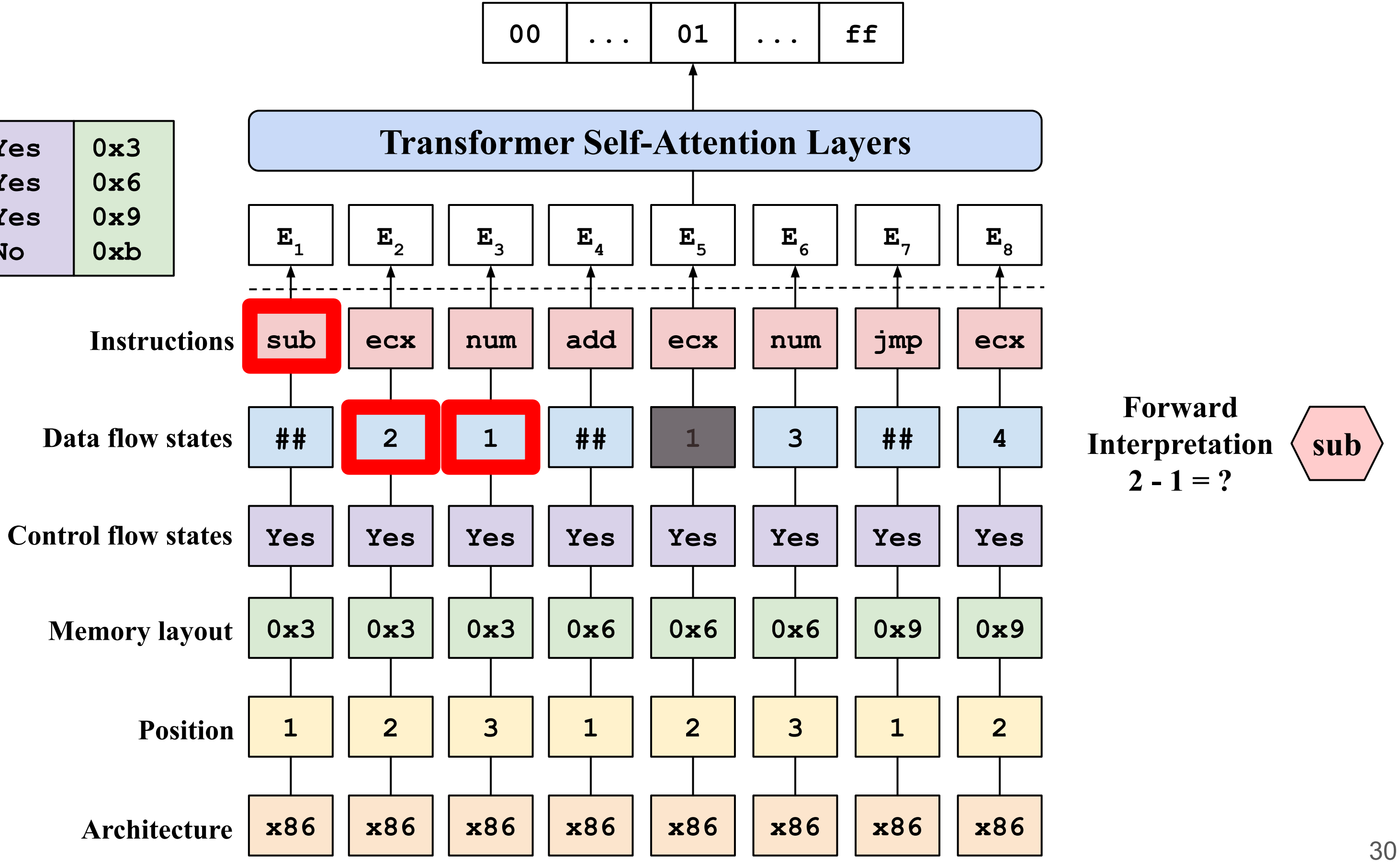
Motivating Example

sub ecx,1	## 2,1	Yes	0x3
add ecx,3	## 1,3	Yes	0x6
jmp ecx	## 4	Yes	0x9
	## 13	No	0xb

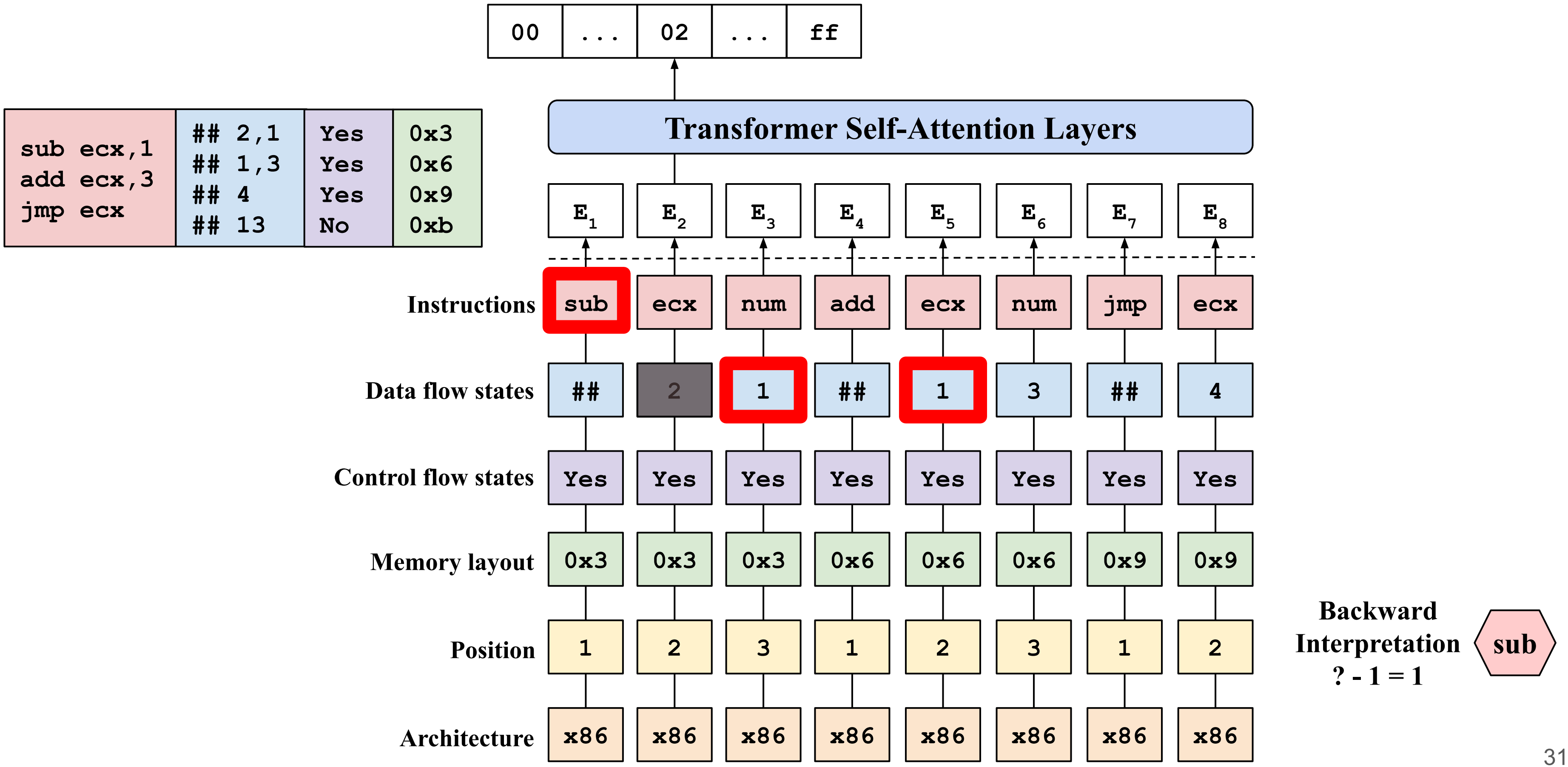


Motivating Example

sub ecx,1
add ecx,3
jmp ecx
	## 2,1
	## 1,3
	## 4
	## 13
		Yes
		Yes
		Yes
		No
			0x3
			0x6
			0x9
			0xb

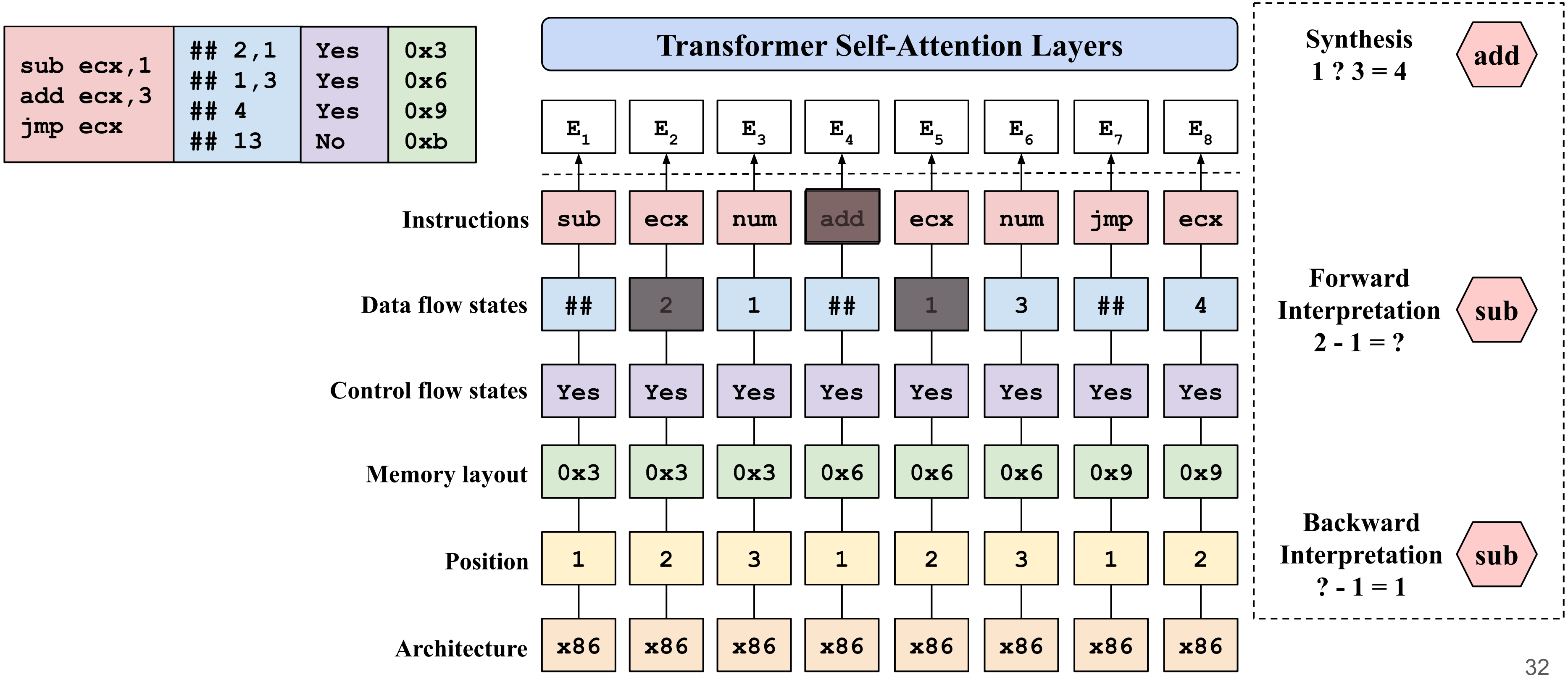


Motivating Example



Motivating Example

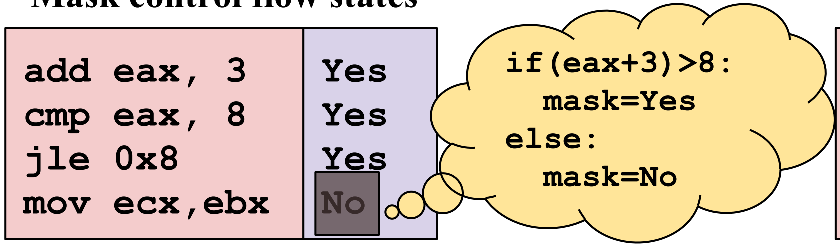
- Program Interpretation
- Program Synthesis



How to train the model to reason about program behavior?

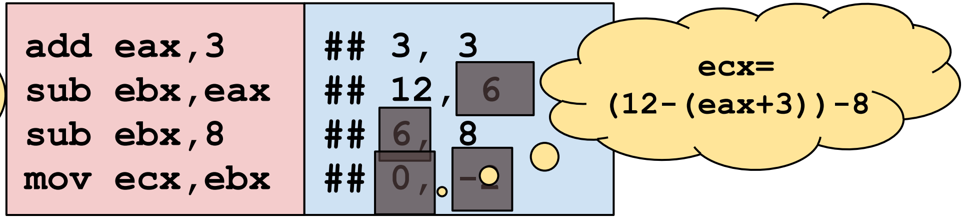
How to train the model to reason about **control flow**?

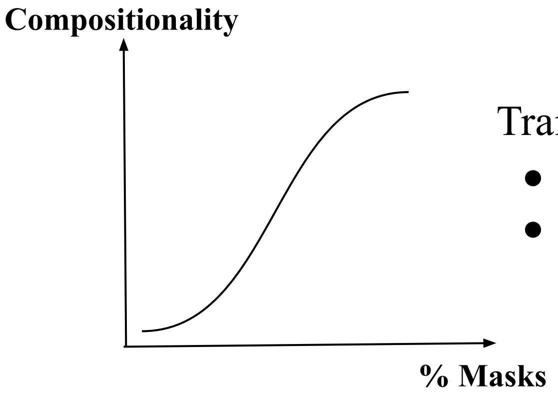
Mask control flow states



How to train the model to reason about instruction **compositions**?

Mask more

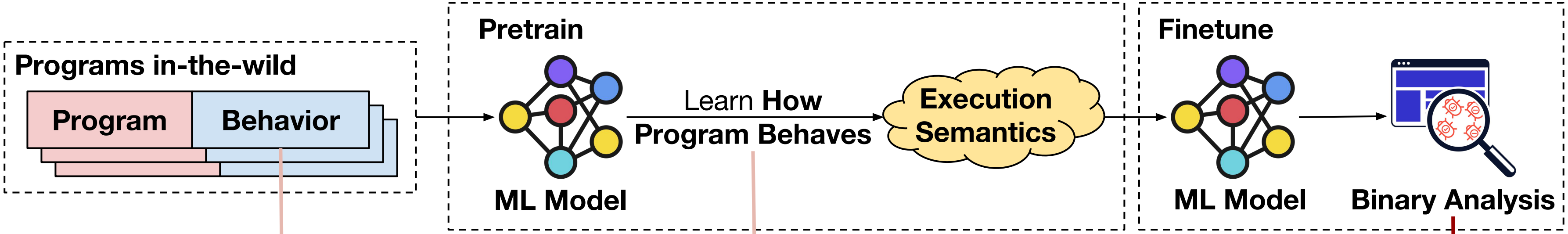




Training

- **Curriculum**: linearly increase % masks by training iterations
- **Randomized** mask selection at each iteration and samples

Challenges of Learning Execution Semantics

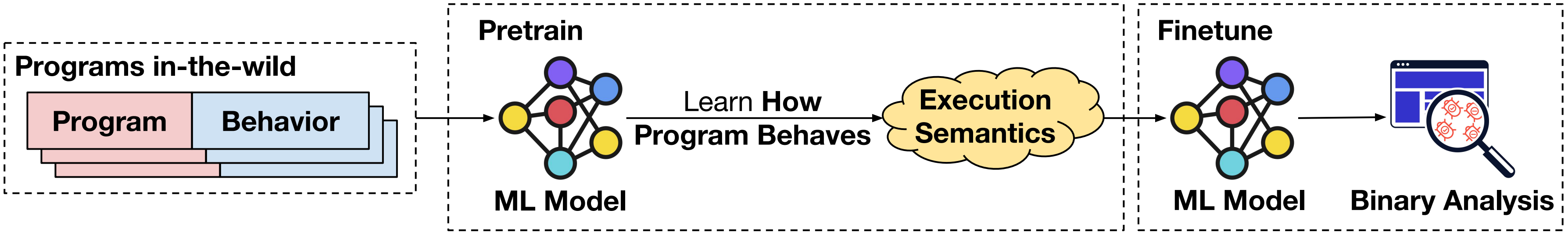


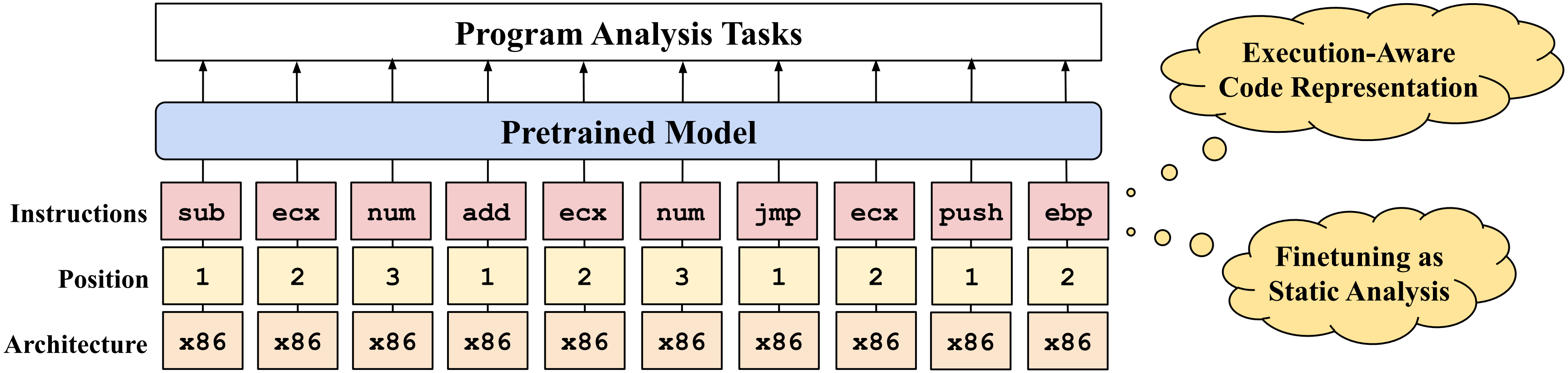
1. How to collect and represent the diverse program behavior?

2. How to train the model to reason about program behavior?

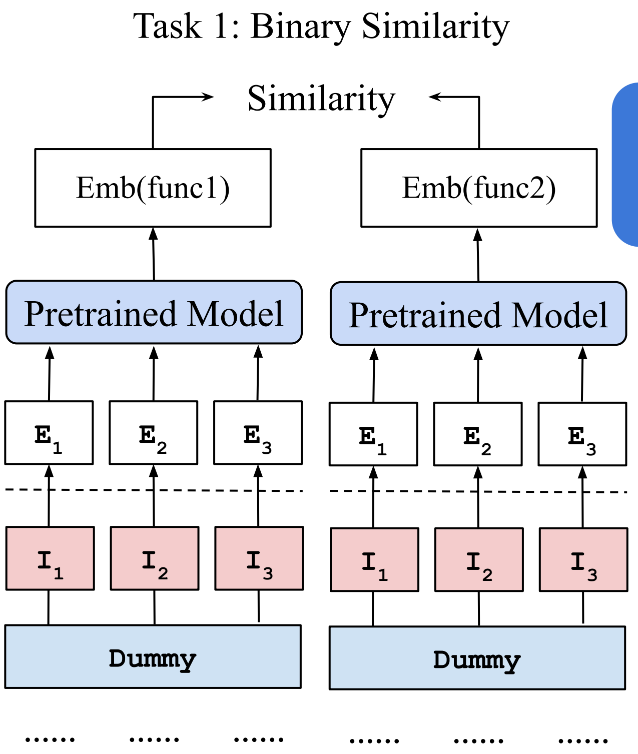
3. How to avoid the expensive dynamic analysis?

How to Avoid the Expensive Dynamic Analysis?

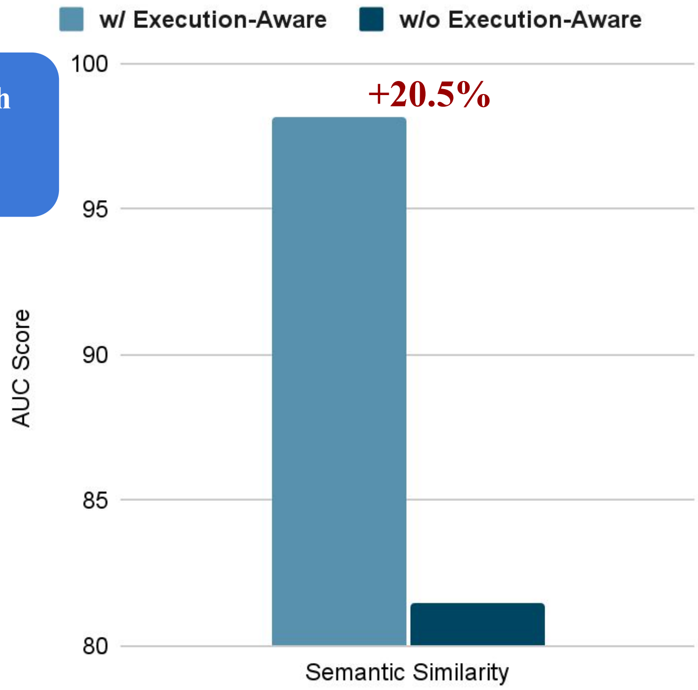




How Much do Learned Execution-Aware Program Representations Help?

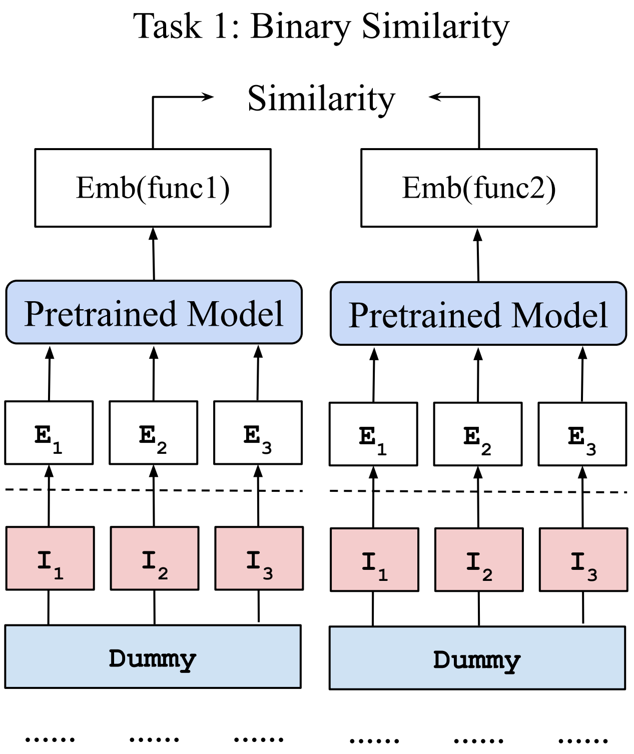


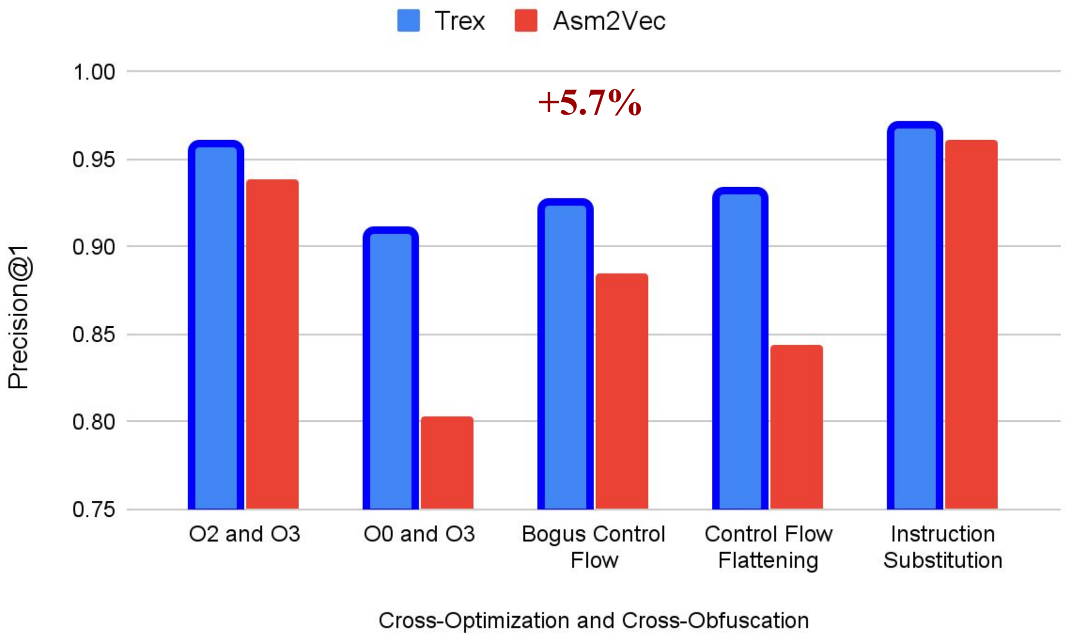
- Vulnerability search
- Malware detection
- Software patching



Pei, Xuan, Yang, Jana, Ray. Trex: Learning Execution Semantics from Micro-traces for Binary Similarity. TSE'22

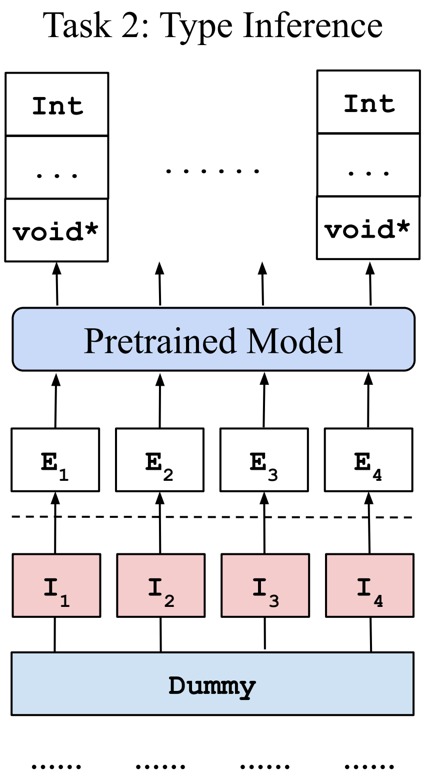
Finetuning for Matching Semantically Similar Binary Functions



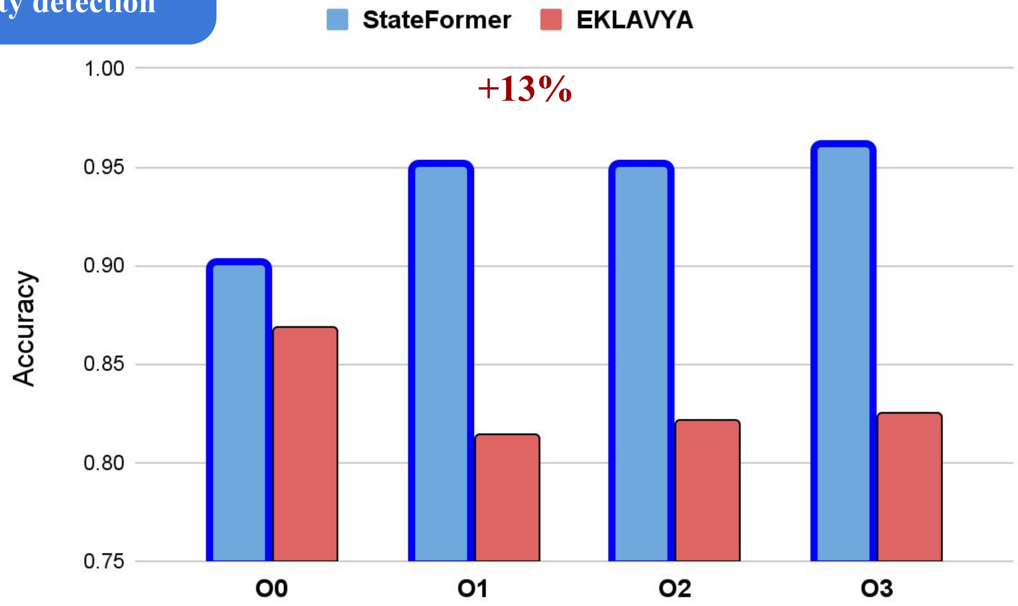


Pei, Xuan, Yang, Jana, Ray. Trex: Learning Execution Semantics from Micro-traces for Binary Similarity. TSE'22

Finetuning for Predicting Function Signatures and Type Inference



- Security retrofitting
- Decompilation
- Vulnerability detection

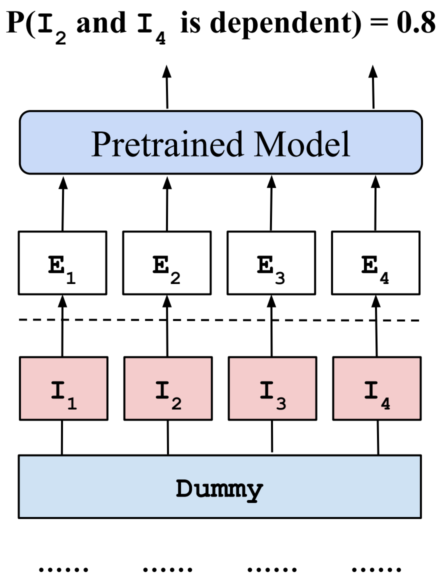


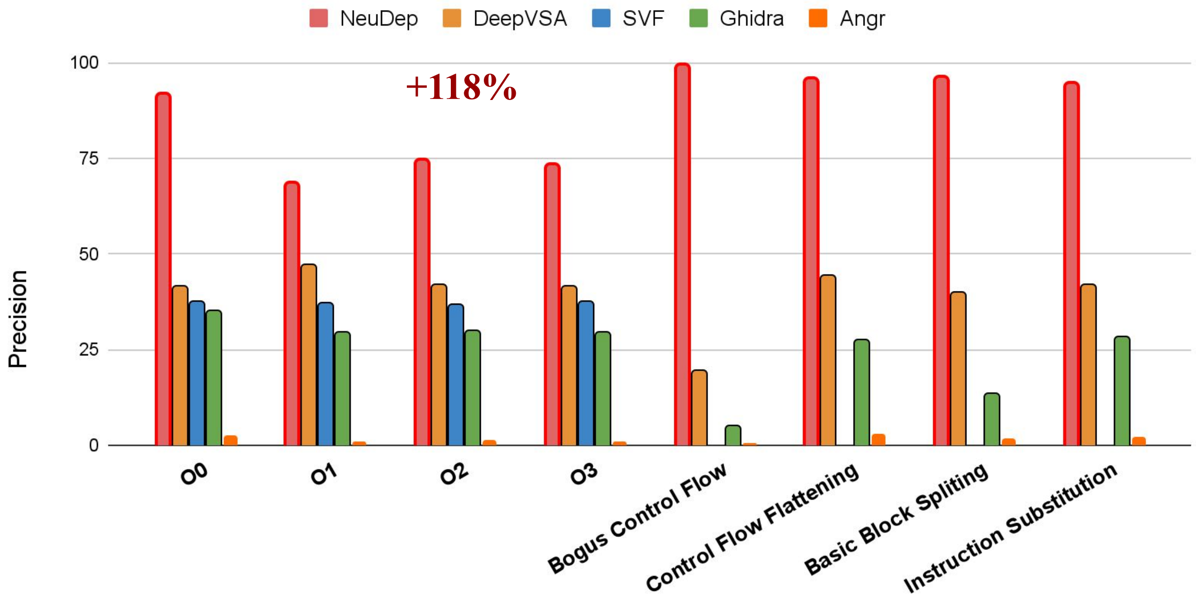
Pei, Guan, Broughton, Chen, Yao, Williams-King, Ummadisetty, Yang, Ray, Jana. StateFormer: Fine-grained Type Recovery from Binaries Using Generative State Modeling. ESEC/FSE'21

Finetuning for Analyzing Memory Dependence

Task 3: Memory Dependence Analysis

- Taint analysis
- Malware Analysis

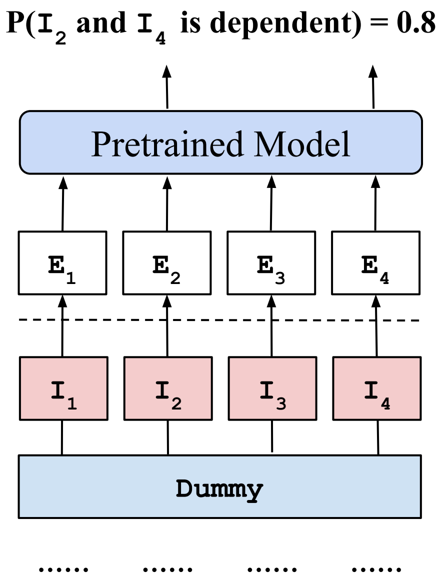


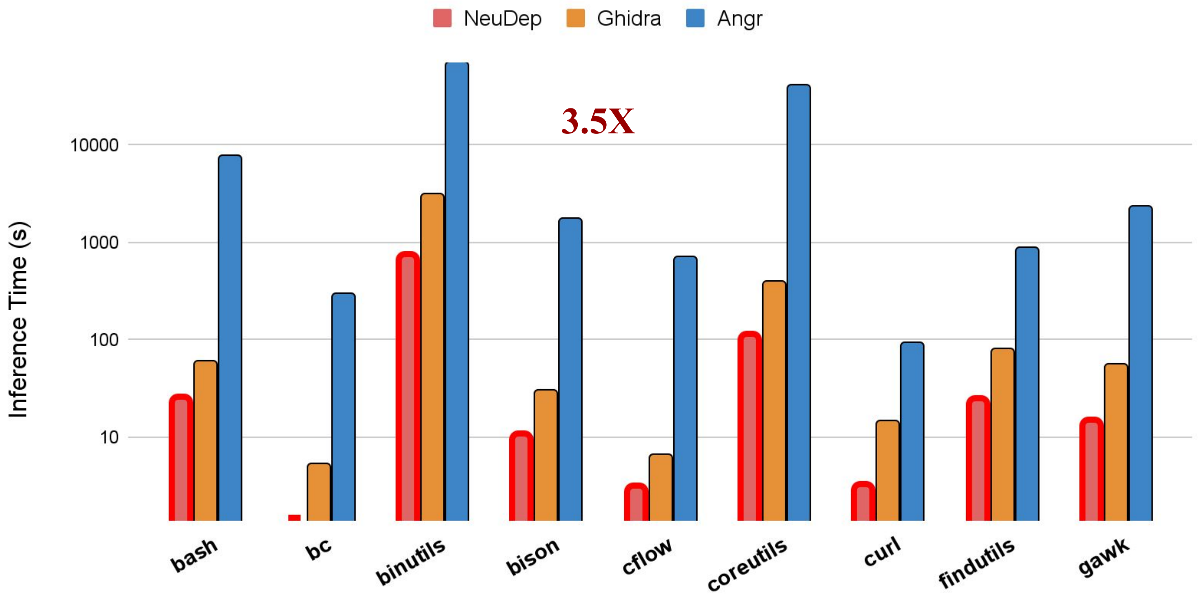


Pei, She, Wang, Geng, Xuan, David, Yang, Jana, Ray. NeuDep: Neural Binary Memory Dependence Analysis. ESEC/FSE’22

Finetuning for Analyzing Memory Dependence: Inference Time

Task 3: Memory Dependence Analysis

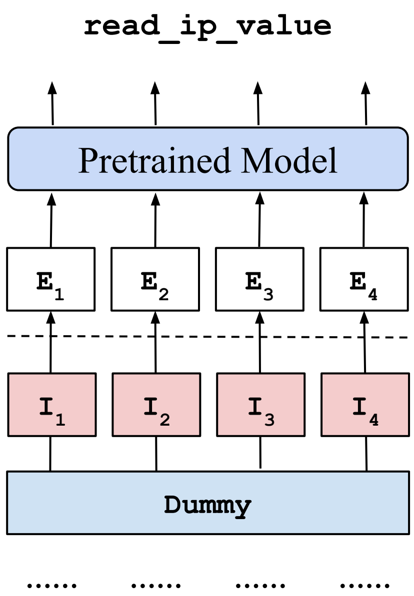




Pei, She, Wang, Geng, Xuan, David, Yang, Jana, Ray. NeuDep: Neural Binary Memory Dependence Analysis. ESEC/FSE'22

Finetuning for Function Name Prediction and Memory Region Prediction

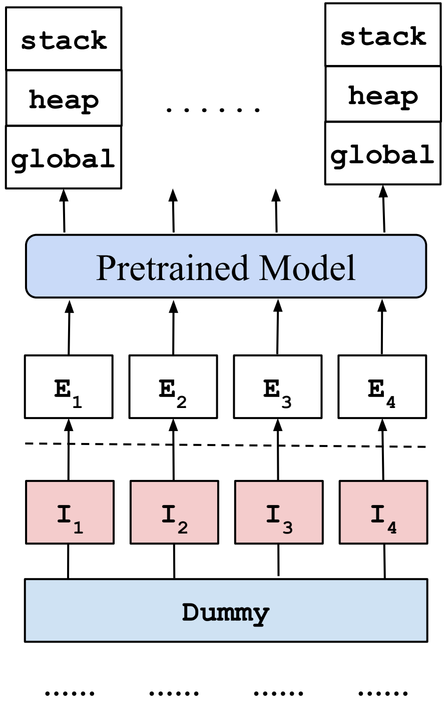
Task 4: Function Name Prediction



- Decompilation
- Reverse engineering

+35% Across All **Architectures** and **Optimizations**

Task 5: Memory Region Prediction



- Value set analysis
- Taint analysis

+7.9% Across All **Types of Memory Regions**

Jin, **Pei**, Wang, Won, Lin. NeuDep: SymLM: Predicting Function Names in Stripped Binaries via Context-Sensitive Execution-Aware Code Embeddings. CCS'22

Pei, She, Wang, Geng, Xuan, David, Yang, Jana, Ray. NeuDep: Neural Binary Memory Dependence Analysis. ESEC/FSE'22

Case Studies: Vulnerability Search in Firmware

CVE	Library	Description
CVE-2019-1563	OpenSSL	Decrypt encrypted message
CVE-2017-16544	BusyBox	Allow executing arbitrary code
CVE-2016-6303	OpenSSL	Integer overflow
CVE-2016-6302	OpenSSL	Allows denial-of-service
CVE-2016-2842	OpenSSL	Allows denial-of-service
CVE-2016-2182	OpenSSL	Allows denial-of-service
CVE-2016-2180	OpenSSL	Out-of-bounds read
CVE-2016-2178	OpenSSL	Leak DSA private key
CVE-2016-2176	OpenSSL	Buffer over-read
CVE-2016-2109	OpenSSL	Allows denial-of-service
CVE-2016-2106	OpenSSL	Integer overflow
CVE-2016-2105	OpenSSL	Integer overflow
CVE-2016-0799	OpenSSL	Out-of-bounds read
CVE-2016-0798	OpenSSL	Allows denial-of-service
CVE-2016-0797	OpenSSL	NULL pointer dereference
CVE-2016-0705	OpenSSL	Memory corruption

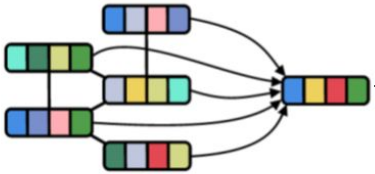
16 Vulnerabilities (Compiled in x86)

Search

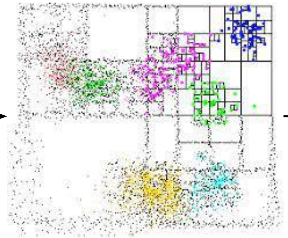
Firmware Images: `Arm`, `MIPS` with unknown compiler flags

Ubiquiti sunMax — TP-Link Deco-M4 — NETGEAR R7000 — Linksys RE7000

15 CVEs — 16 CVEs — 7 CVEs — 14 CVEs



Learned Function Embeddings

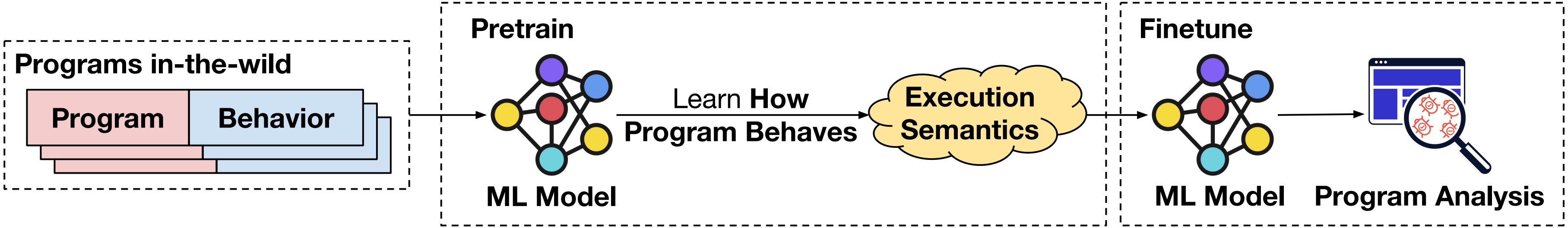


Approximate Nearest Neighbors

Search over **1.4M** functions within **6.3 seconds**

Pei, Xuan, Yang, Jana, Ray. Trex: Learning Execution Semantics from Micro-traces for Binary Similarity. TSE'22

Summary: Learning **Program Semantics** via **Execution-Aware Pre-training** Improves Program Analysis



Precise: Outperforms the state-of-the-art by up to **118%**

Efficient: Speedup over the off-the-shelf tool by up to **98.1x**

Generalizable and Robust: Remains accurate across

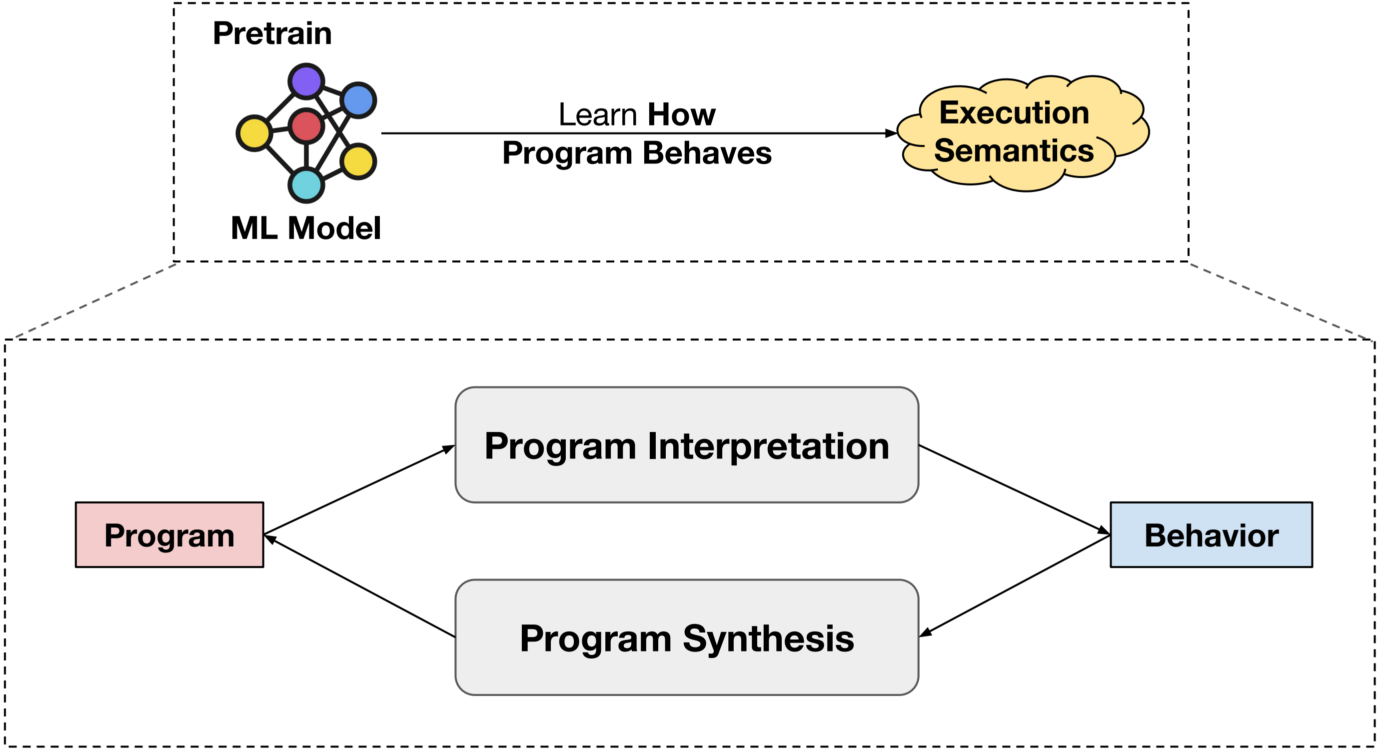
Visual C++

x64 x86

-O0 -O1 -O2
-O3 -Od Ox

Compilers | Architectures | Optimizations | Obfuscations

Limitation: Learning Execution-Aware Program Representations is Challenging



Extremely challenging to learn precise semantics

Limitation: What the Model has Learned during Pretraining?

Instructions	Dataflow states
`......`	`......`
`sub ecx,0x3`	`## 0x5,0x3`
`add ecx,0x4`	`## 0x2,0x4`
`......`	`......`

Instructions	Dataflow states
`......`	`......`
`sub ecx,0x3`	`## 0x43,0x3`
`add ecx,0x4`	`## 0x3d,0x4`
`......`	`......`

Perturb dataflow states from 0x5 to 0x43

Ground-truth	Top-1	Top-2
`0x2`	`0x2` (98%)	`0x3` (2%)
`0x3d`	`0x3a` (28%)	`0x33` (13%)

Does not extrapolate well

Exciting Future Work

Democratize the development of secure software system via

Data-Driven Program Analysis

Multi-Modal Software

Learn, analyze, and interact with heterogeneous software modalities?

Security Applications

Automate existing security tasks and enable new security applications?

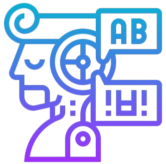
Robustness and Generalization

Develop and evaluate to ensure strong robustness guarantee?

Software is Inherently Heterogeneous and Multi-Modal

Noisy

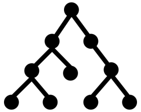
Physical Interaction

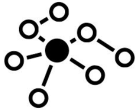


Natural Language

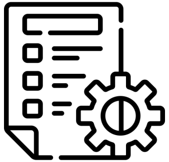
```
"""
  Returns the year and
quarter of an input date.
  Args:
    date: XXXX-XX-XX
  Returns:
    year: XXXX
    quarter: XXX
"""
```

Software Program



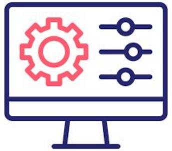


```
while i < n
   invariant  i<= n
{
   i = i + 1 ;
}
```



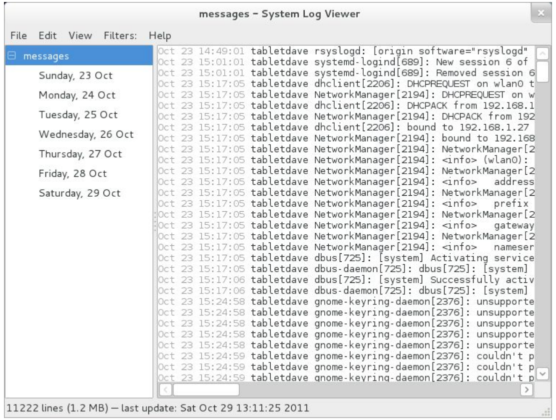
Specifications

```
port=81
rootpath=...
execution=True
verification=True
......
```



Configurations

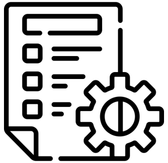
Program Activity



Abundant

How to learn from heterogeneous software modalities?

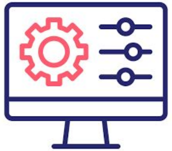

Physical Interaction


Specifications

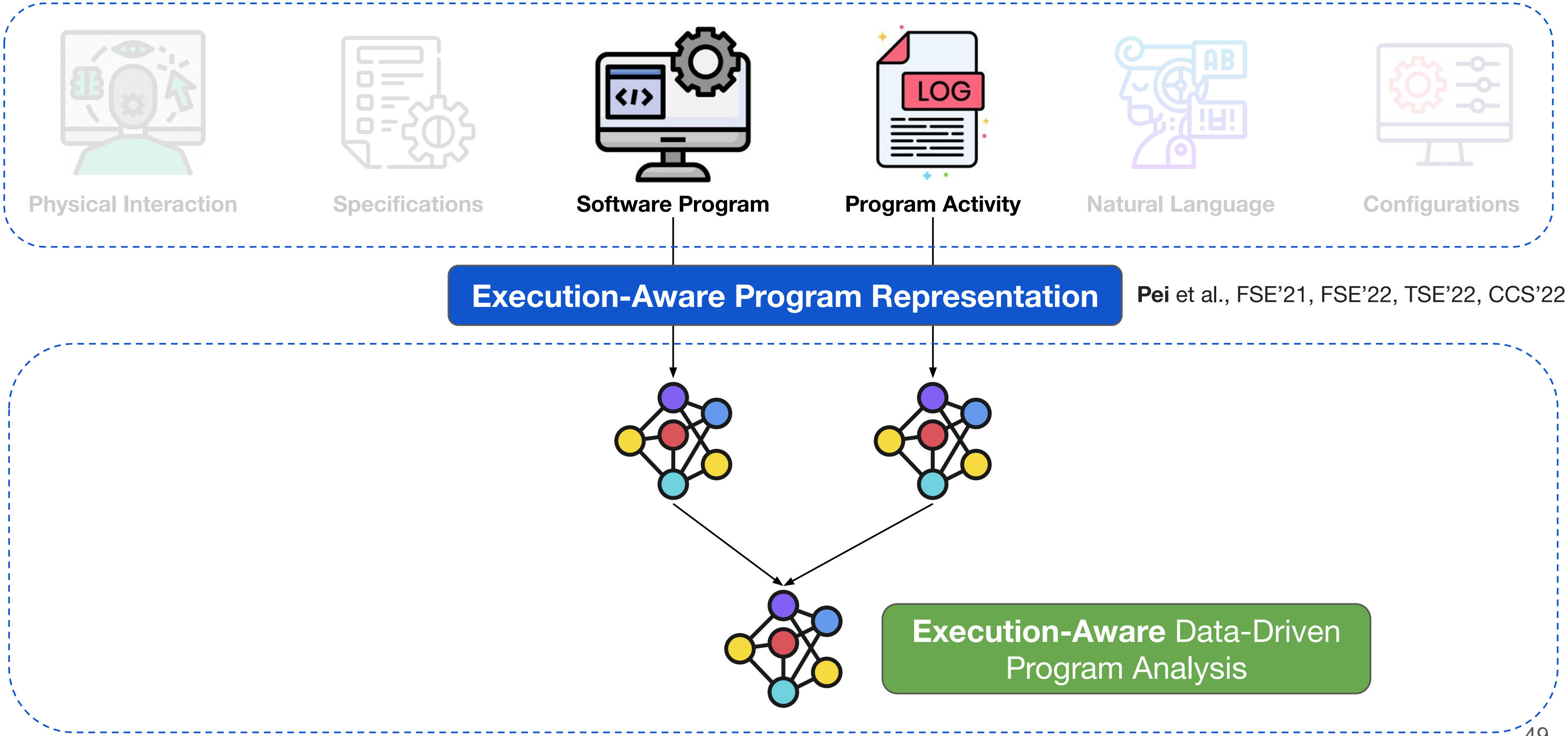

Software Program

Program Activity

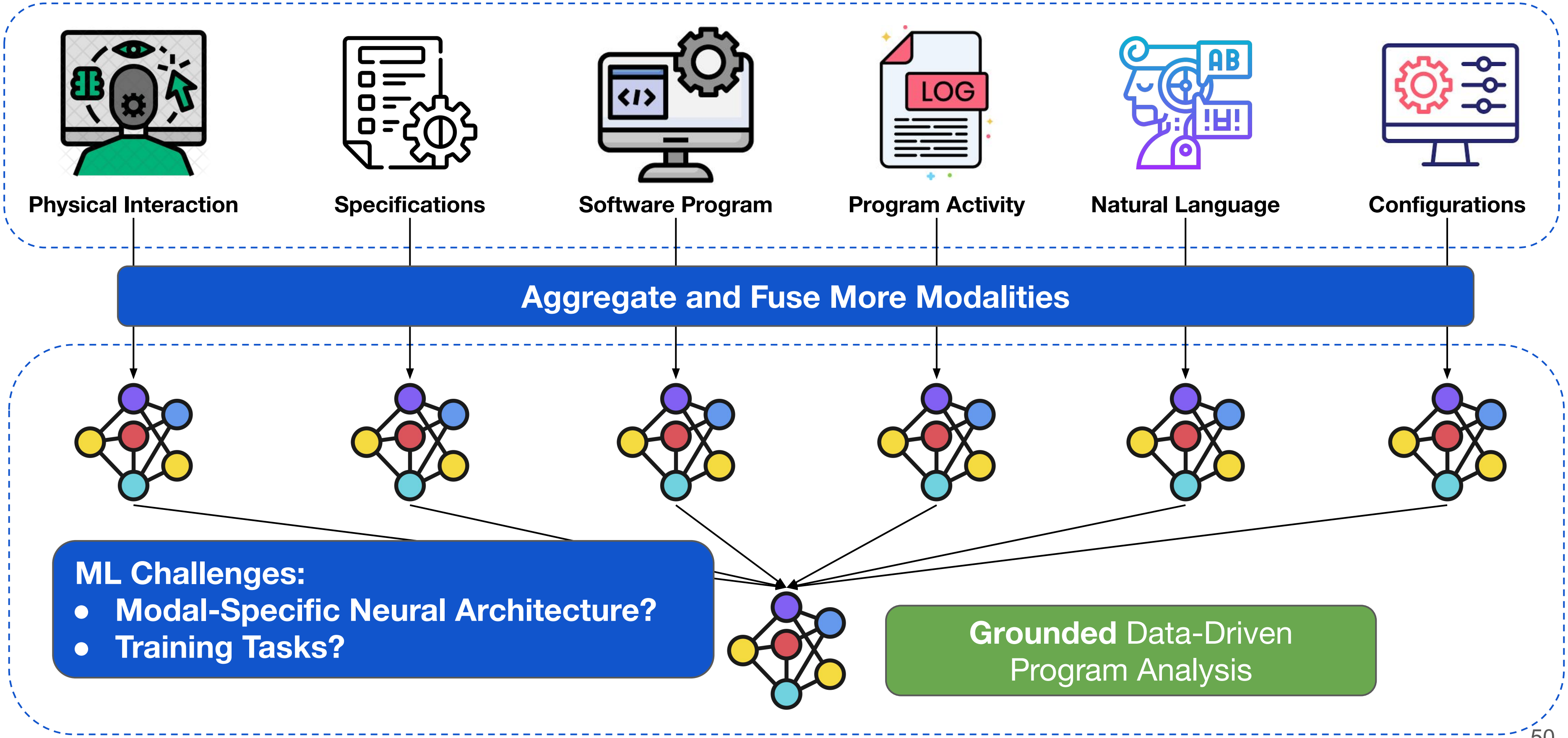
Natural Language


Configurations

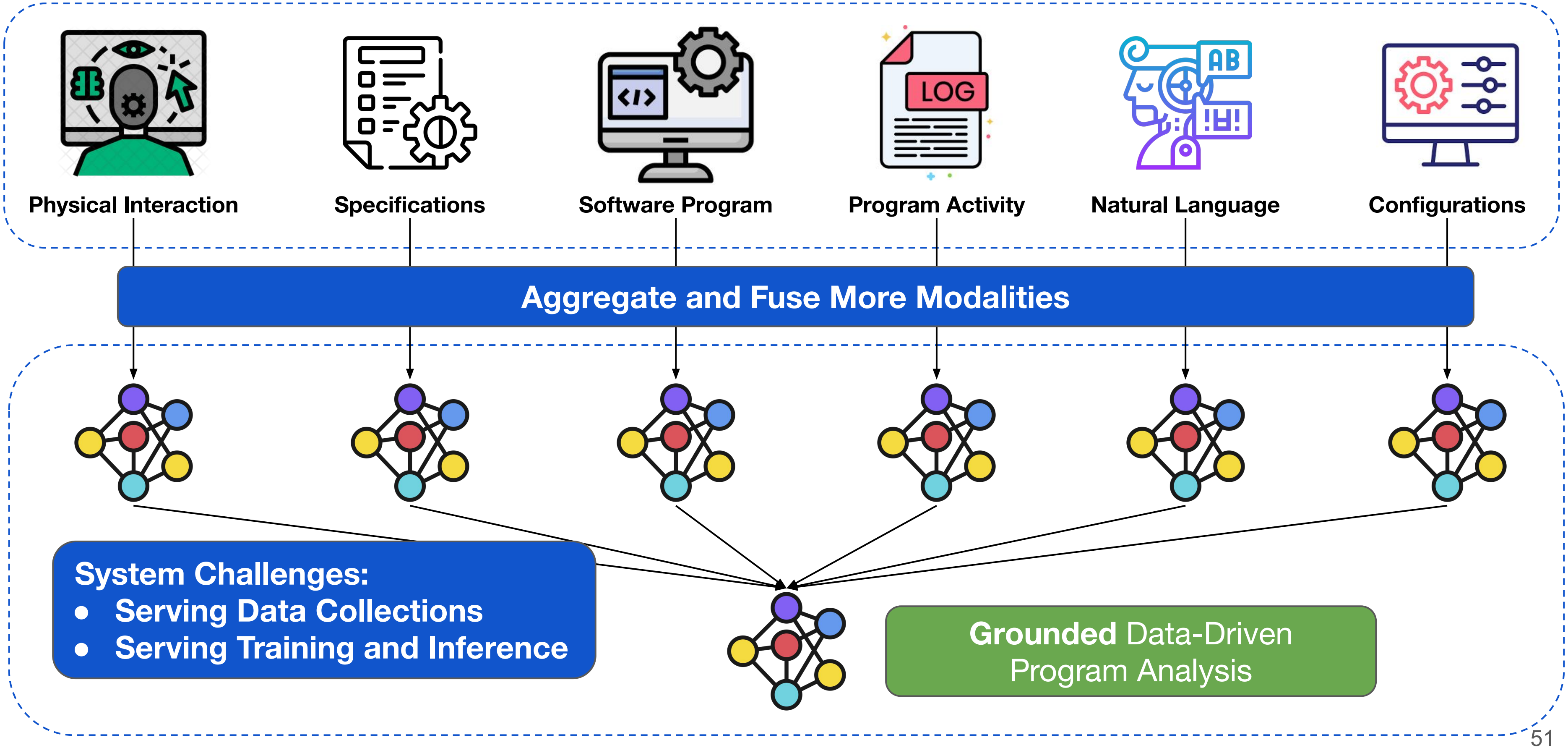
How to learn from heterogeneous software modalities?



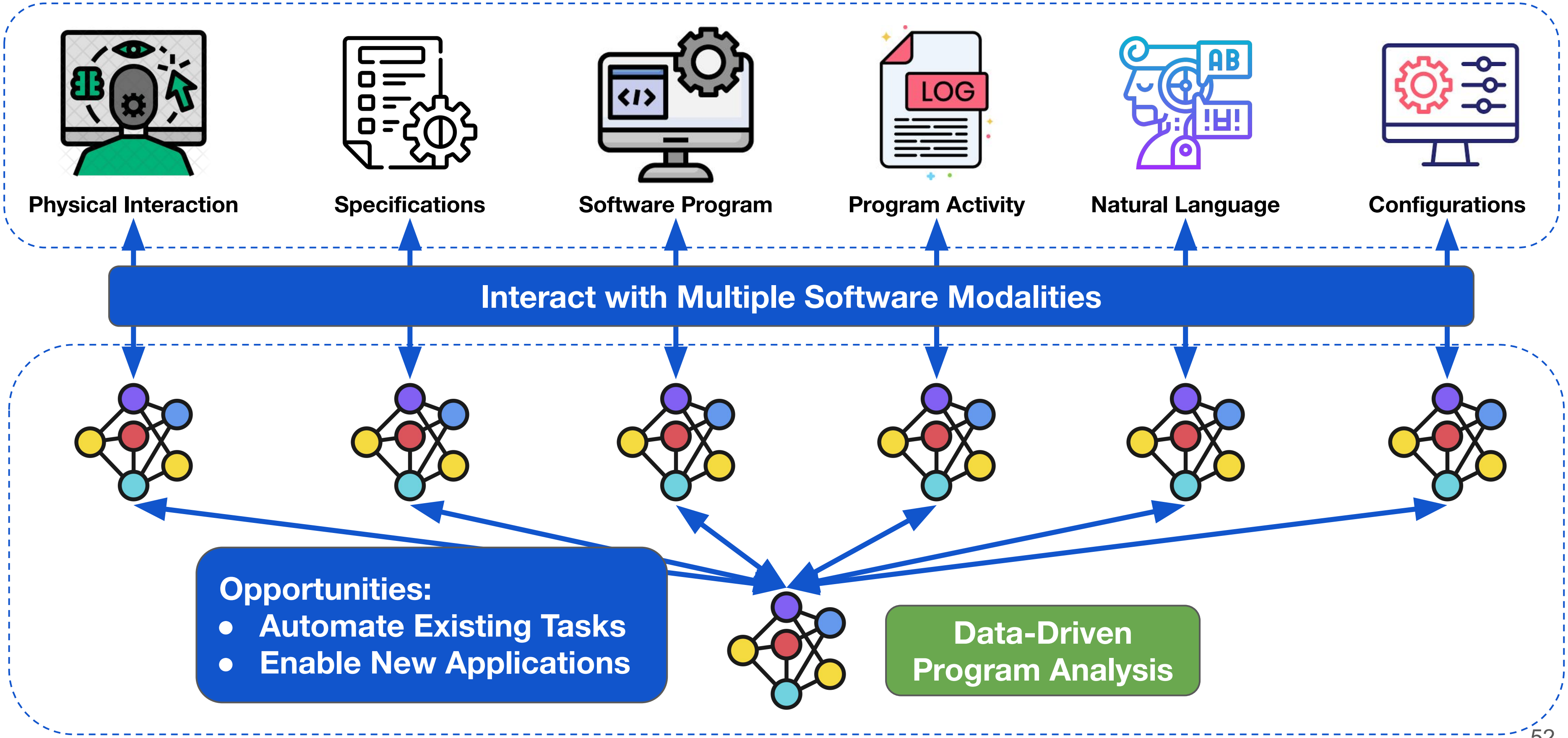
How to learn from heterogeneous software modalities?



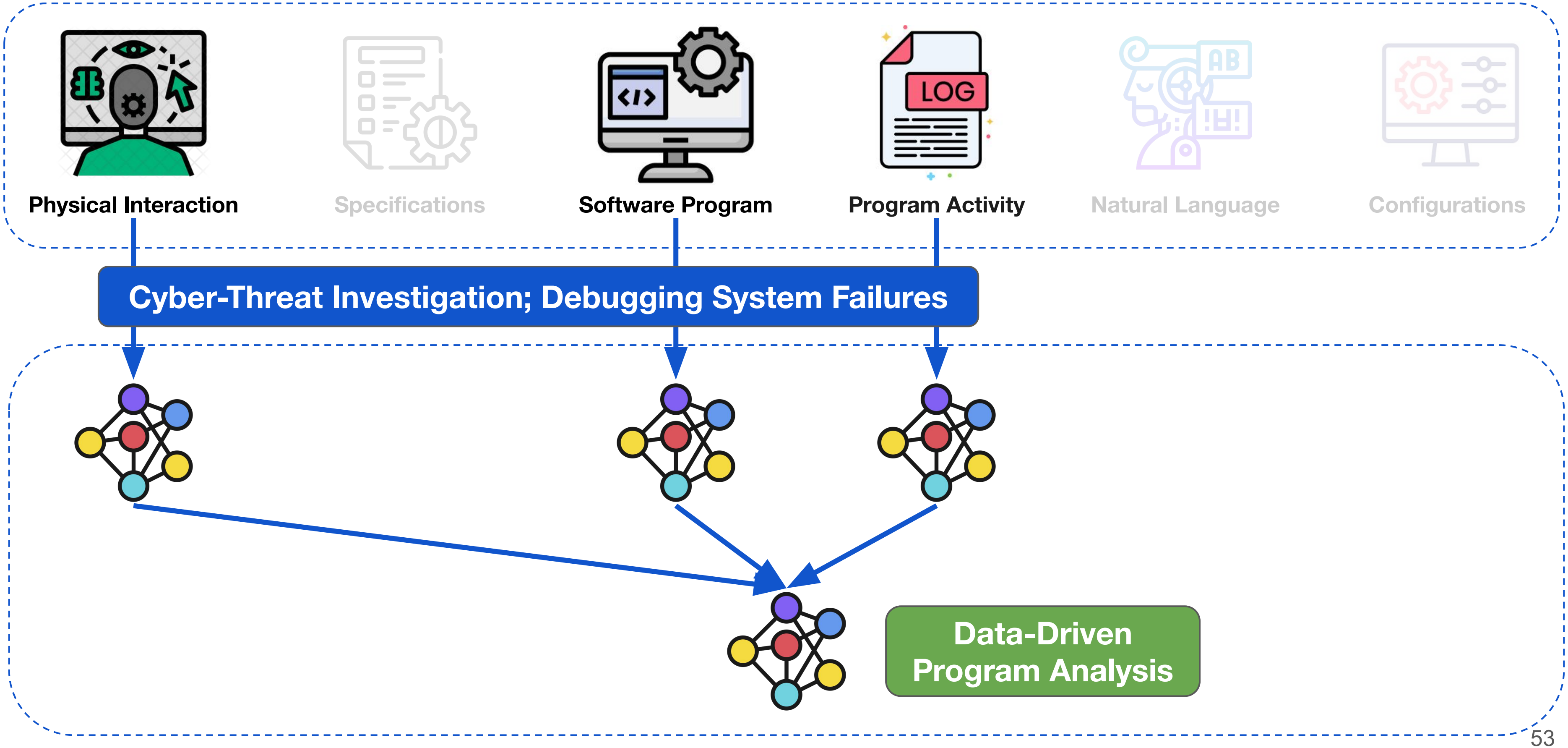
How to learn from heterogeneous software modalities?



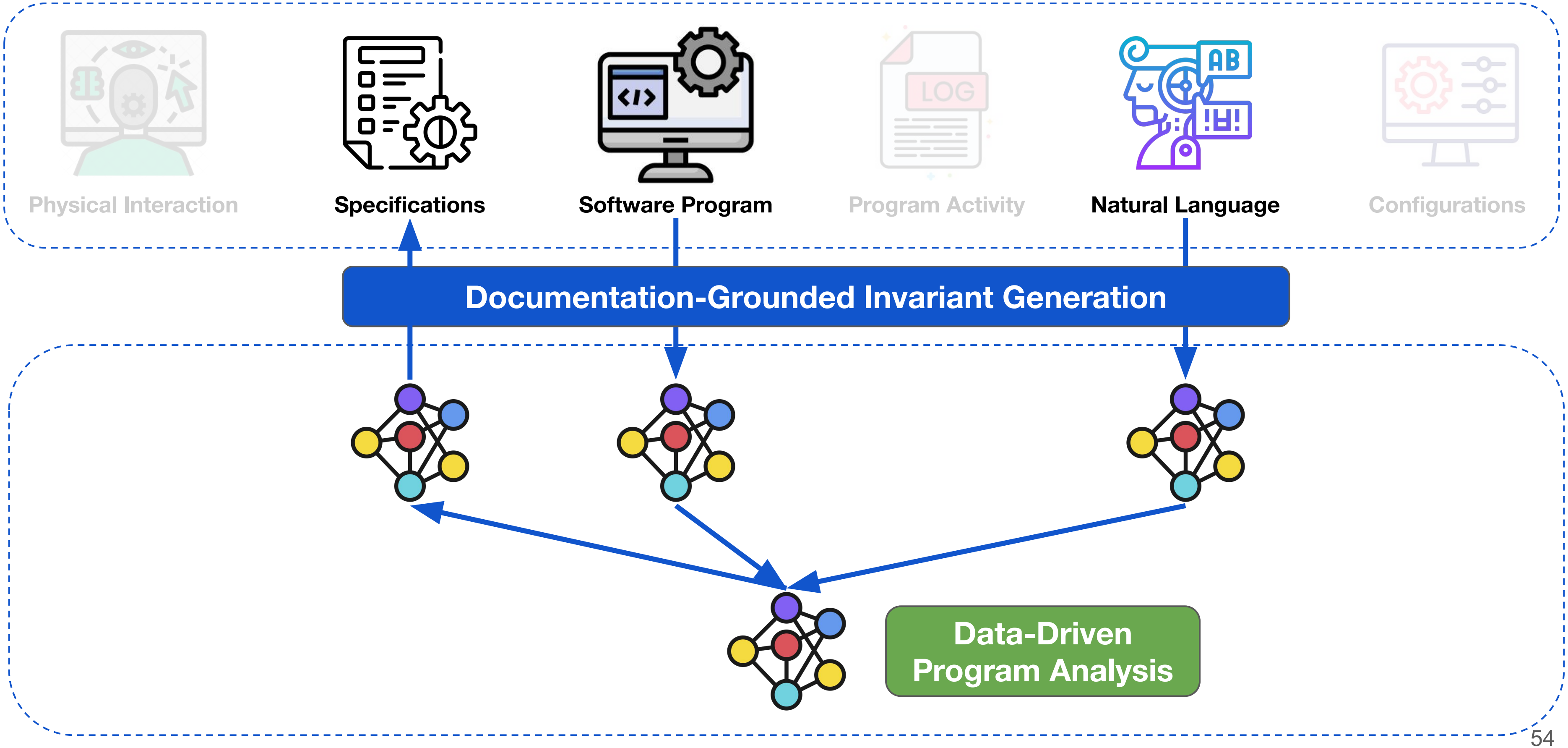
How to interact with heterogeneous software modalities?



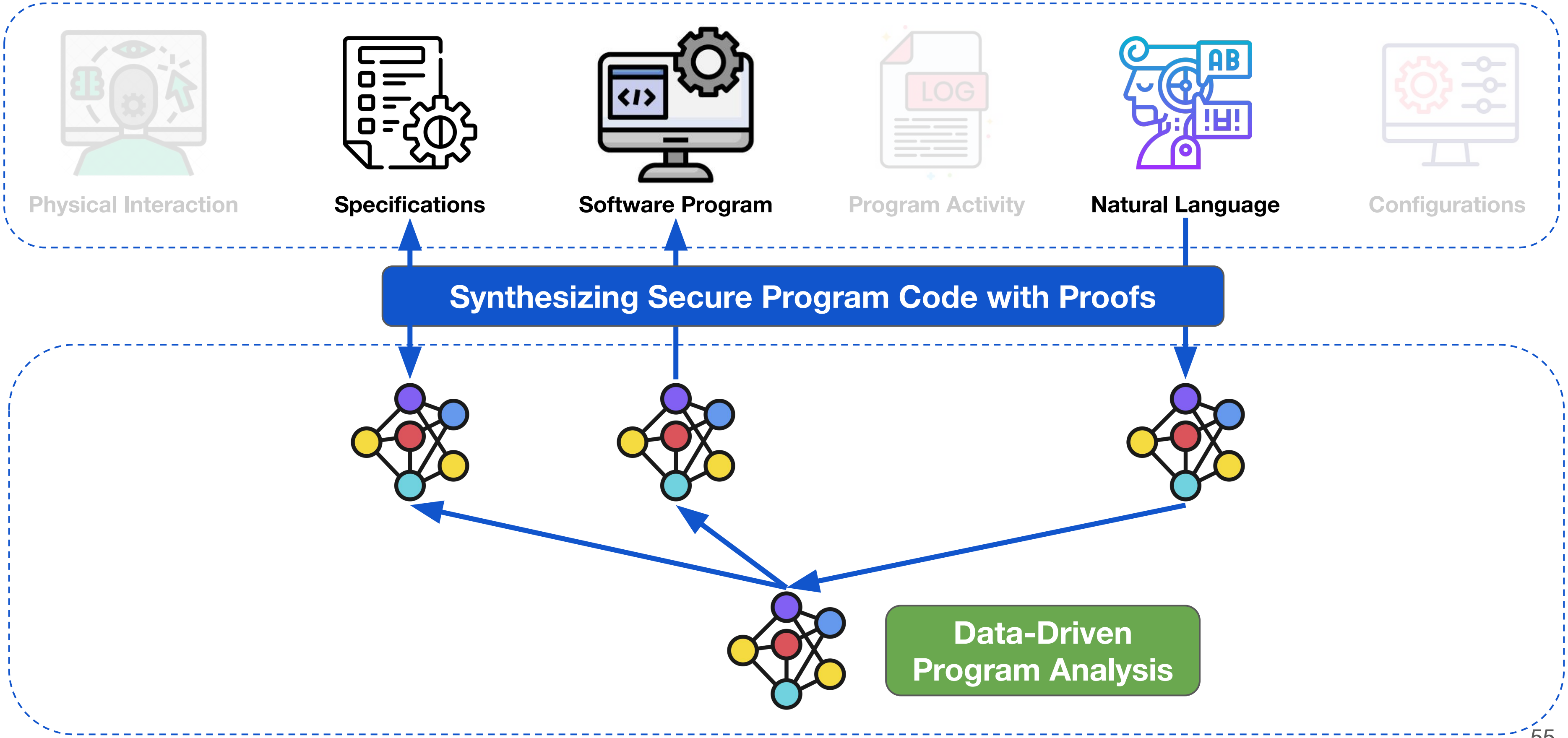
Automating Existing Security Applications



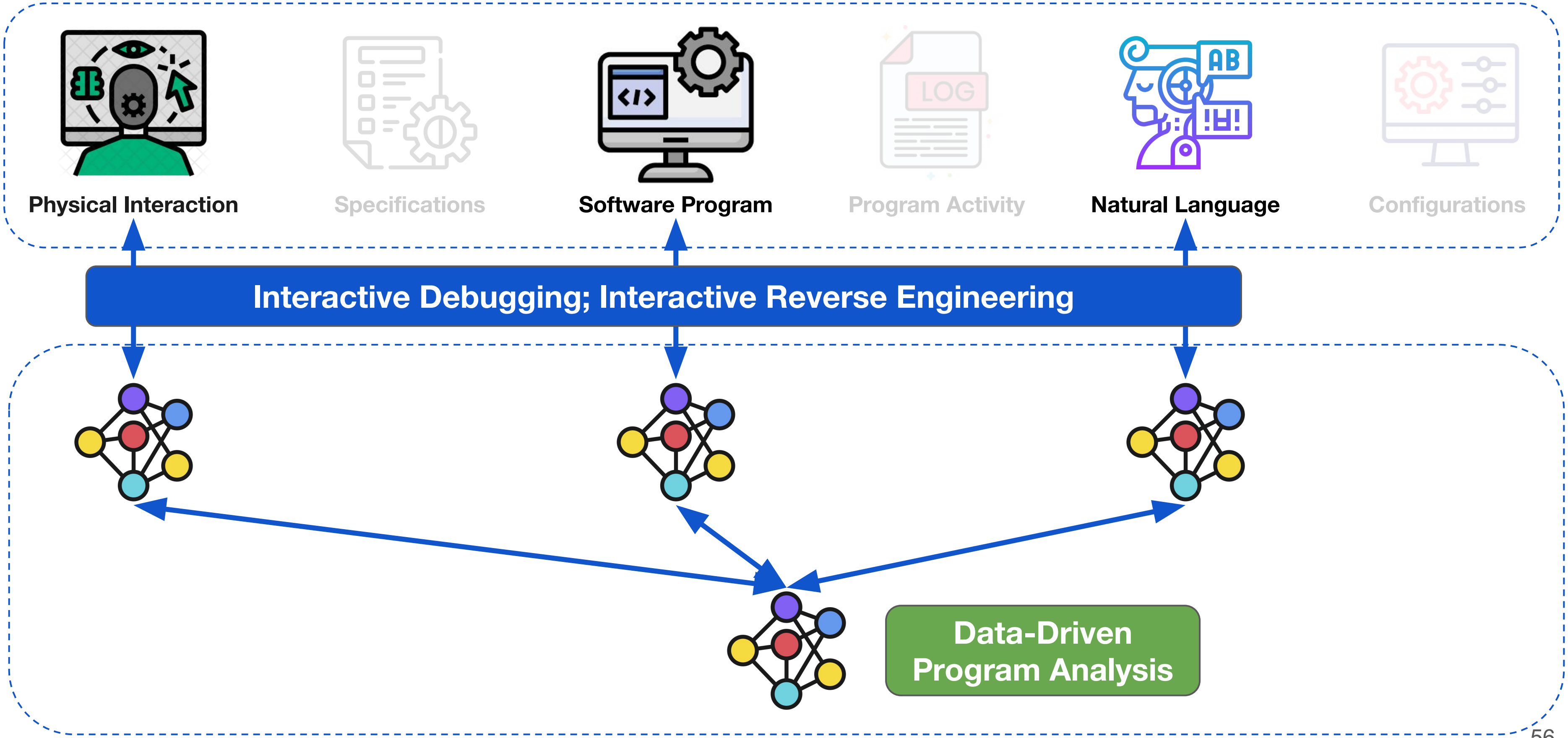
Enabling New Security Applications



Enabling New Security Applications

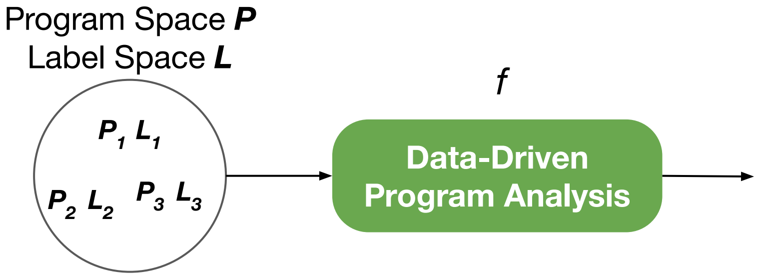


Enabling New Security Applications



Principled Robustness Measurement

Current testing of data-driven program analysis: **Random Testing**

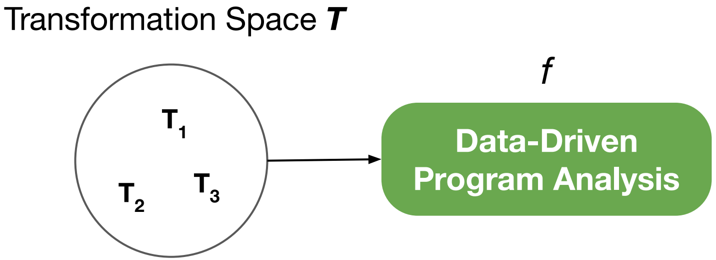


Evaluate

$$f(\boldsymbol{P_1}) = \boldsymbol{L_1}$$
$$f(\boldsymbol{P_2}) = \boldsymbol{L_2}$$
$$f(\boldsymbol{P_3}) = \boldsymbol{L_3}$$

How to systematically explore the program space?

Future testing of data-driven program analysis: **Transformation-Oriented Testing**



T alters program syntax: Robustness Testing

$$f(T_1(\boldsymbol{P})) = f(\boldsymbol{P})$$
$$f(T_2(\boldsymbol{P})) = f(\boldsymbol{P})$$
$$f(T_3(\boldsymbol{P})) = f(\boldsymbol{P})$$

Systematic Testing and Verification of Neural Networks

Systematic Whitebox Testing
[SOSP'17 **Best Paper Award**, ICSE'18]

Formal Verification of Security Properties
[Usenix'18, Neurips'18, DeepTest'18]

Visual Transformations

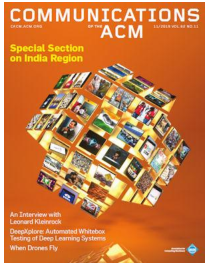
CSAW'18 APPLIED RESEARCH COMPETITION

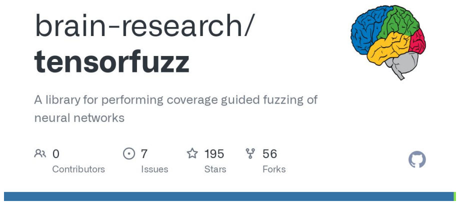
IEEE SPECTRUM

techradar.

Newsweek

SCIENTIFIC AMERICAN

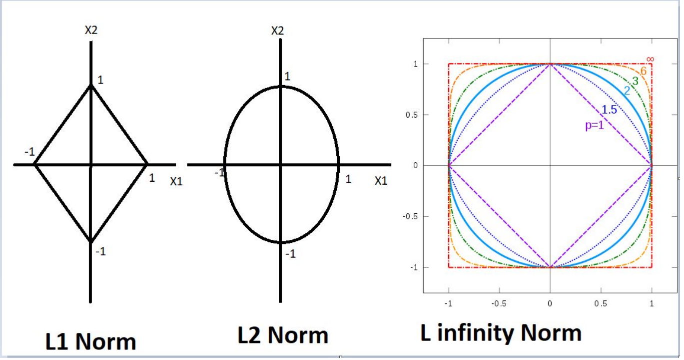




Data-Driven Program Analysis

$$T = \begin{array}{ccc} \text{Translation} & \text{Scaling} & \text{Aspect ratio change} \\ \begin{bmatrix} 1, & 0, & dx \\ 0, & 1, & dy \end{bmatrix} & \begin{bmatrix} s, & 0, & 0 \\ 0, & s, & 0 \end{bmatrix} & \begin{bmatrix} r, & 0, & 0 \\ 0, & 1, & 0 \end{bmatrix} \\ \text{Shear} & \text{Reflect} & \text{Rotation} \\ \begin{bmatrix} 1, & c, & 0 \\ d, & 1, & 0 \end{bmatrix} & \begin{bmatrix} -1(1), & 0, & 0 \\ 0, & 1(-1), & 0 \end{bmatrix} & \begin{bmatrix} \cos(\theta), -\sin(\theta), & 0 \\ \sin(\theta), \cos(\theta), & 0 \end{bmatrix} \end{array}$$

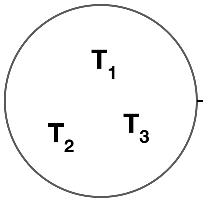
$T =$



Formal verification of all possible transformations

Data-Driven Program Analysis with Provable Robustness by Construction

Transformation Space ***T***



Robust ***f*** by Construction

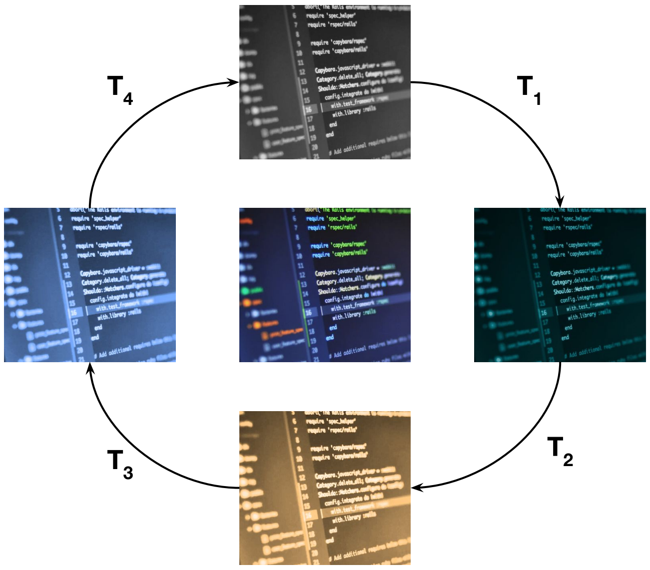
Data-Driven Program Analysis

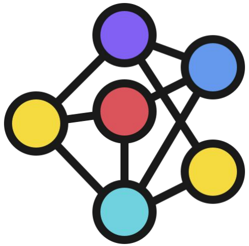
$$f(\boldsymbol{T_i}(\boldsymbol{P})) = f(\boldsymbol{P}) \quad \forall\, T_i \in \boldsymbol{T}$$

Permutation Group *T*

```
1:  x=5                                1:  y=6
2:  y=6      Instruction Reordering    2:  x=5
3:  z=x+y                              3:  z=x+y
```

.
.
.





Symmetry-Preserving Model Architectures e.g., self-attention, graph NN

Thanks!