# CS5733 Program Synthesis

## #22. Neural and NS Synthesis : LLMs + Synthesis

Ashish Mishra, November 22, 2024

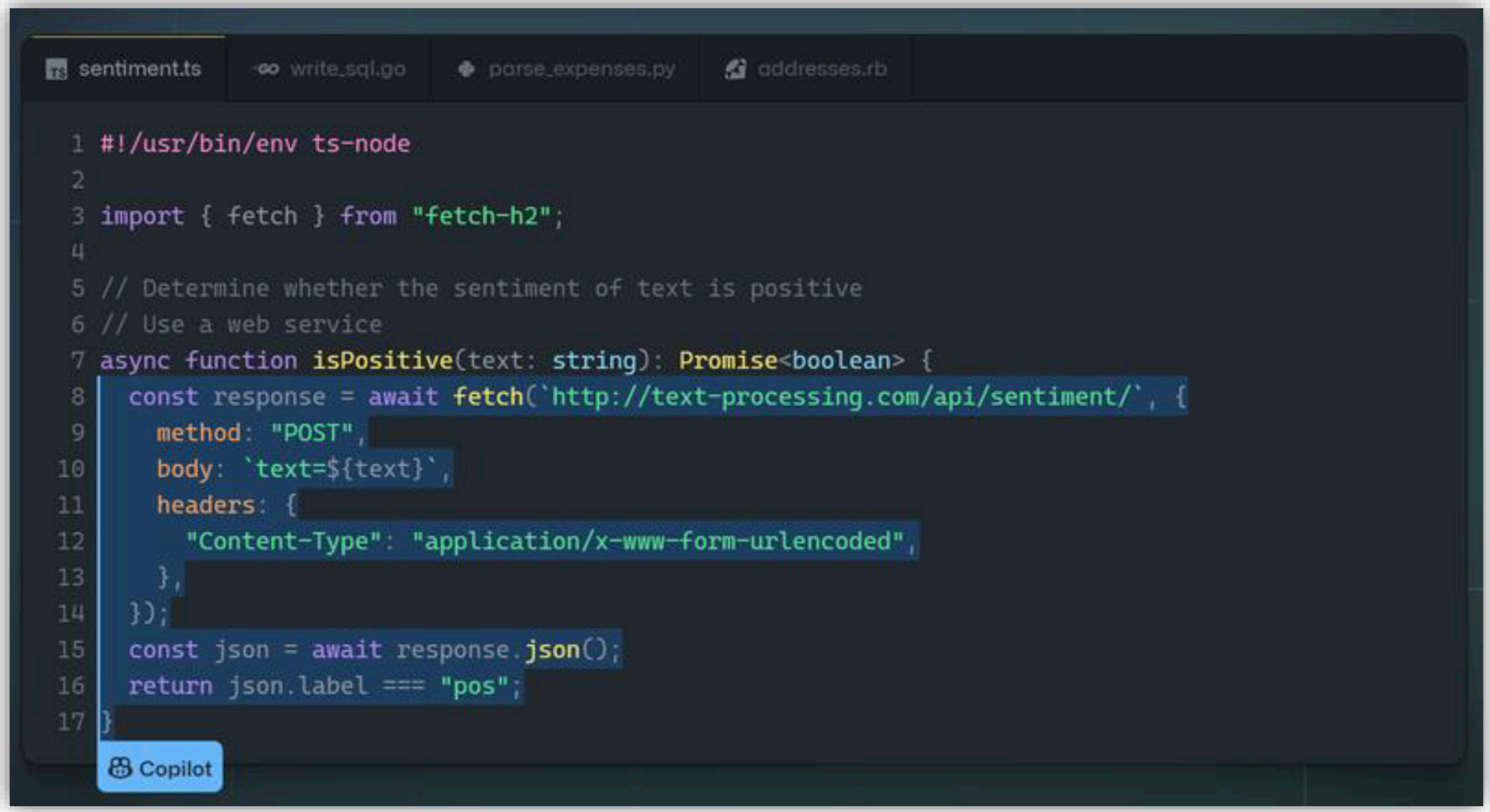
With material from Nadia Polikarpova and Armando
-Solar

# Final Class for the course

# Plan for the class

- Tomorrow : LLM Era
  - synthesis from natural language
  - how can we make LLMs generate better code?

# LLMs 4 Code




aws

CodeWhisperer

# …but they are not perfect

according to a survey of 410 developers [Liang et al, ICSE’24]:

- the most popular reason developers don’t use LLMs is that generated code **“doesn’t meet functional or non-functional (e.g., security, performance) requirements that I need”**

according to [Perry et al, CCS’23]:

- participants with an AI assistant wrote significantly less secure code
- and were more likely to believe that they wrote secure code!

# Two challenges

Accuracy

LLMs provide no guarantees that spec is satisfied

How do we increase the probability that a generated program matches user intent?

Validation

Spec is partly informal: NL, code context

How do we determine if a program matches user intent?

# Techniques

Accuracy

Constrained Decoding

Fine Tuning

Validation

Self-consistency

User interaction

High-level DSL

# Techniques

Accuracy

Constrained Decoding

Fine Tuning

Validation

Self-consistency

User interaction

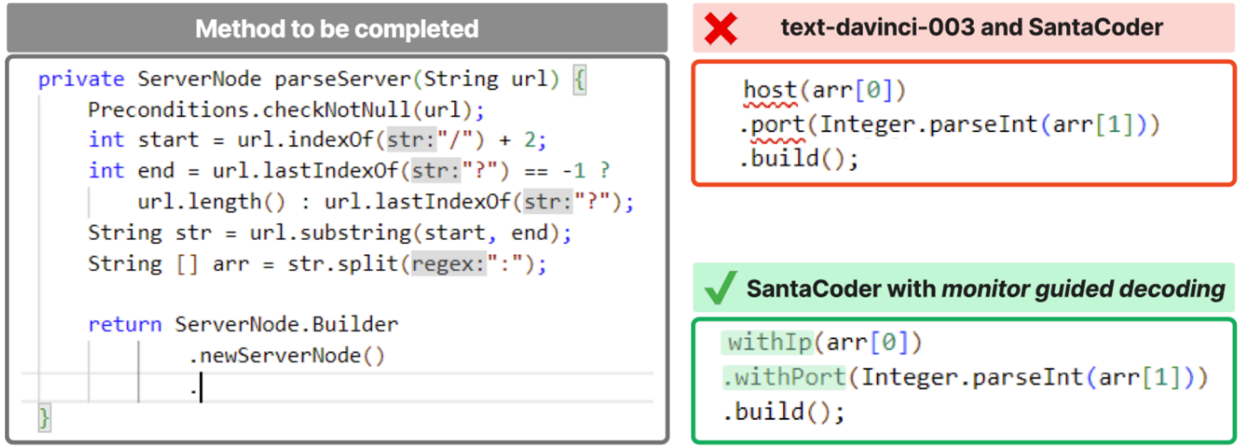
High-level DSL

# Monitor-guided Decoding

[Agrawal et al: *Monitor-guided decoding of code LMs with static analysis of repository context.* NeurIPS’23]

LLMs struggle to produce correct code in the context of a repo

*Idea:* use a language server to mask LLM token predictions



# Problem

LMs suffer from **limited awareness of repository-level context** (e.g., files and dependencies) – especially in private settings and not seen during training

Hence, LMs end up using types defined in other files incorrectly, for example, hallucinating undefined names at dereference locations

**Method to be completed**

```java
private ServerNode parseServer(String url) {
    Preconditions.checkNotNull(url);
    int start = url.indexOf(str:"/") + 2;
    int end = url.lastIndexOf(str:"?") == -1 ?
        url.length() : url.lastIndexOf(str:"?");
    String str = url.substring(start, end);
    String [] arr = str.split(regex:":");

    return ServerNode.Builder
            .newServerNode()
            .
}
```

**text-davinci-003 and SantaCoder**

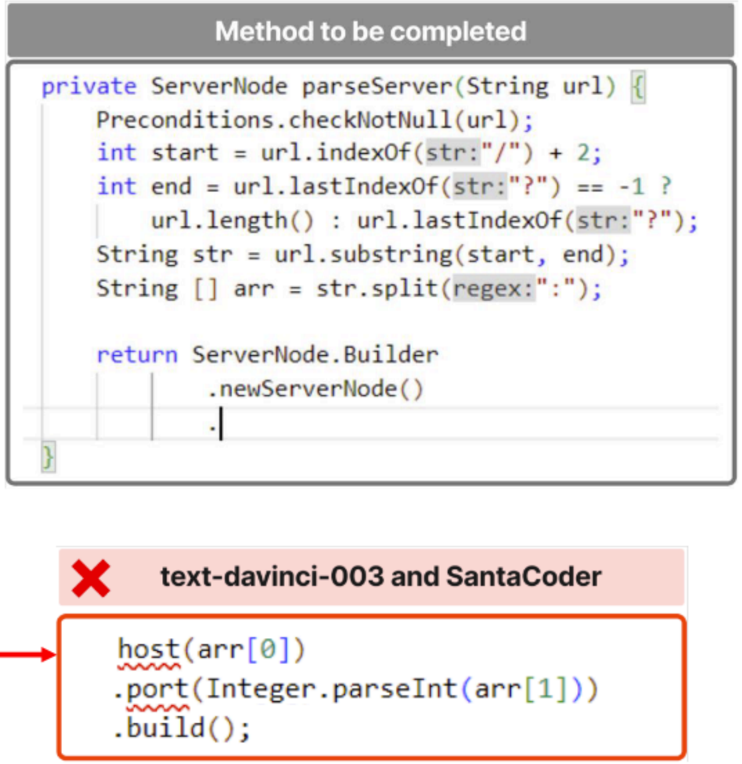
```java
host(arr[0])
.port(Integer.parseInt(arr[1]))
.build();
```

# Problem

LMs suffer from **limited awareness of repository-level context** (e.g., files and dependencies) – especially in private settings and not seen during training

Hence, LMs end up using types defined in other files incorrectly, for example, hallucinating undefined names at dereference locations

Recent techniques use retrieval-based prompting, which bloats up the context, and is limited by LM context window size. If the prompts do not have all the relevant information, the LMs still end up hallucinating.



# Monitor Guided Decoding

**Method to be completed**

```java
private ServerNode parseServer(String url) {
    Preconditions.checkNotNull(url);
    int start = url.indexOf(str:"/") + 2;
    int end = url.lastIndexOf(str:"?") == -1 ?
        url.length() : url.lastIndexOf(str:"?");
    String str = url.substring(start, end);
    String [] arr = str.split(regex:":");

    return ServerNode.Builder
            .newServerNode()
            .
}
```

**text-davinci-003 and SantaCoder**

```java
host(arr[0])
.port(Integer.parseInt(arr[1]))
.build();
```

**SantaCoder with *monitor guided decoding***

```java
withIp(arr[0])
.withPort(Integer.parseInt(arr[1]))
.build();
```

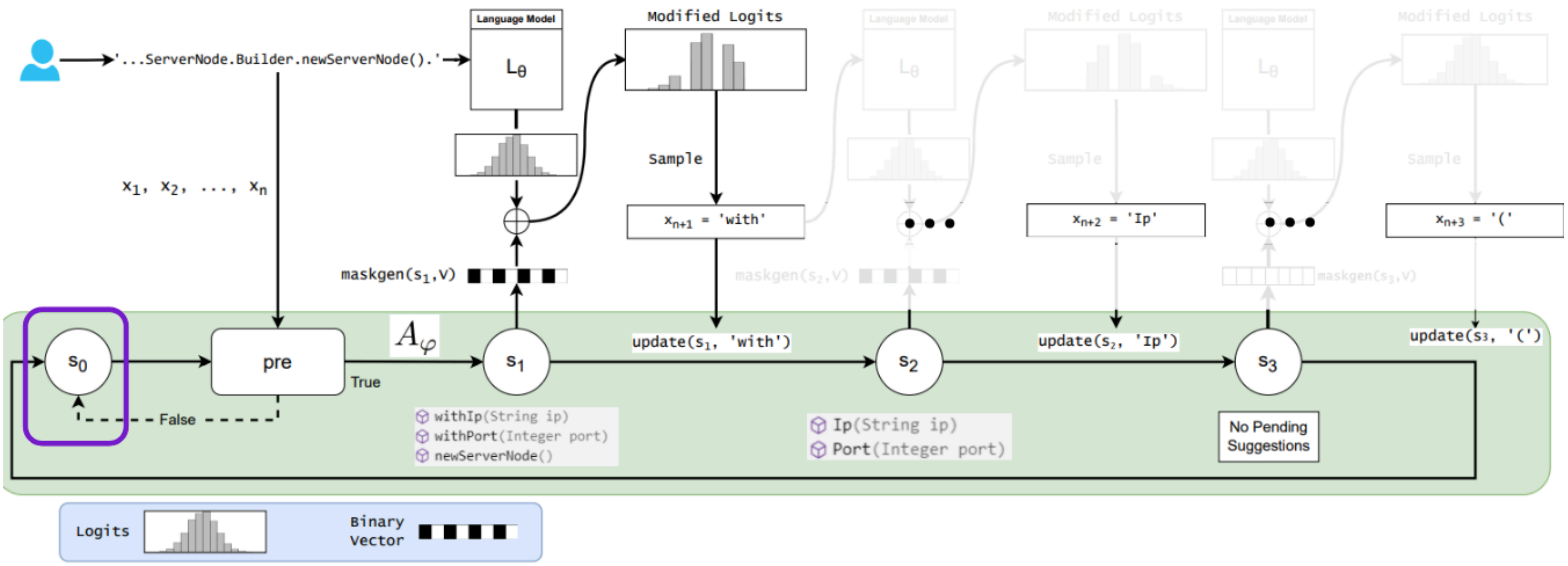
**Intuition:** IDEs assist human developers by providing global context information during code authoring.
We extend this IDE assistance to LMs.

***Monitor guided decoding (MGD)*** defines *monitor* as a stateful interface between LMs and static analysis.

A *monitor* runs concurrently to the decoder. It iteratively uses results from continuous static analysis to mask tokens inconsistent with the static analysis.

MGD is a generalizable technique that works across programming languages, coding scenarios and can use many different static analyses for monitoring

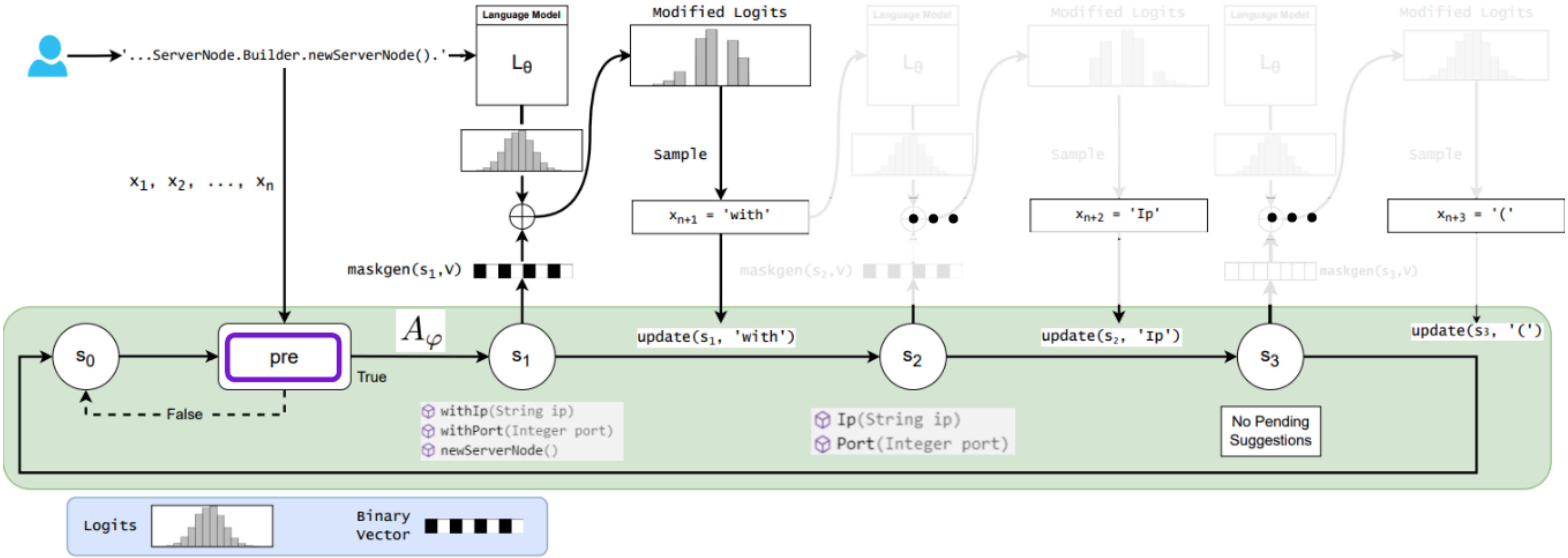
# Working of Monitor Guided Decoding




$s_0$ $s_1$ $s_2$ ...

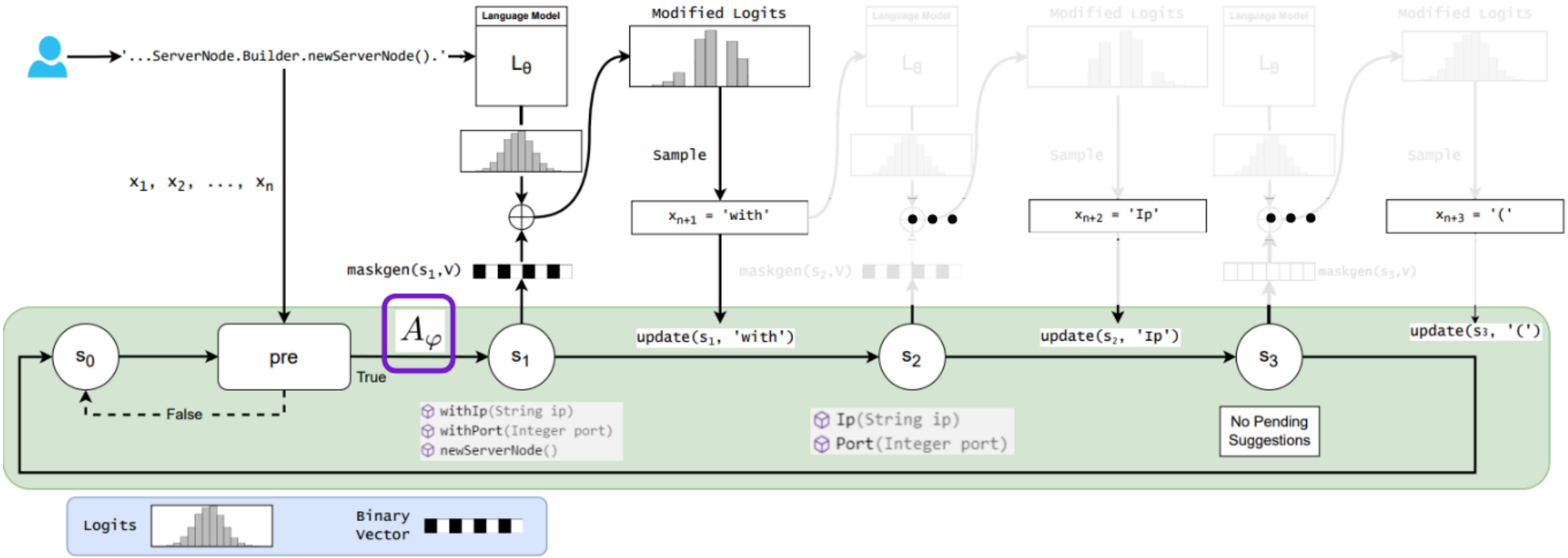
$s_0$ is the default state in which all vocabulary tokens are valid. All the other states represent constraints to be applied for the next token.

# Working of Monitor Guided Decoding



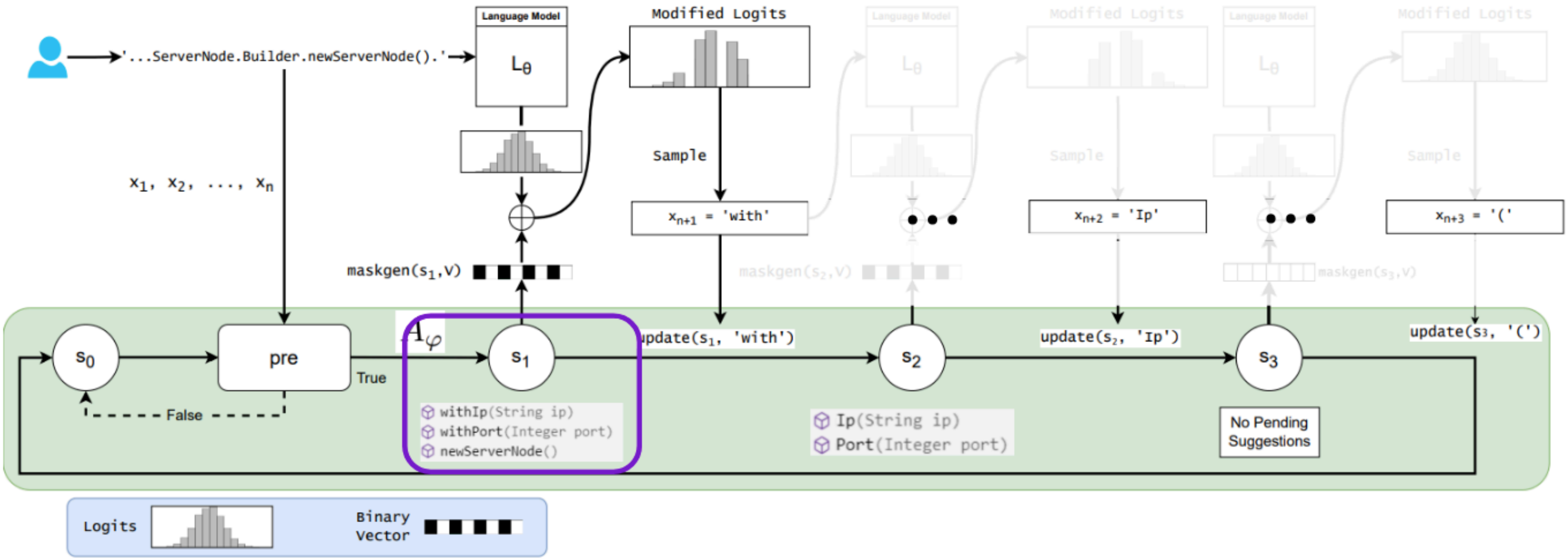

pre Precondition check – determines when to trigger the static analysis.

# Working of Monitor Guided Decoding

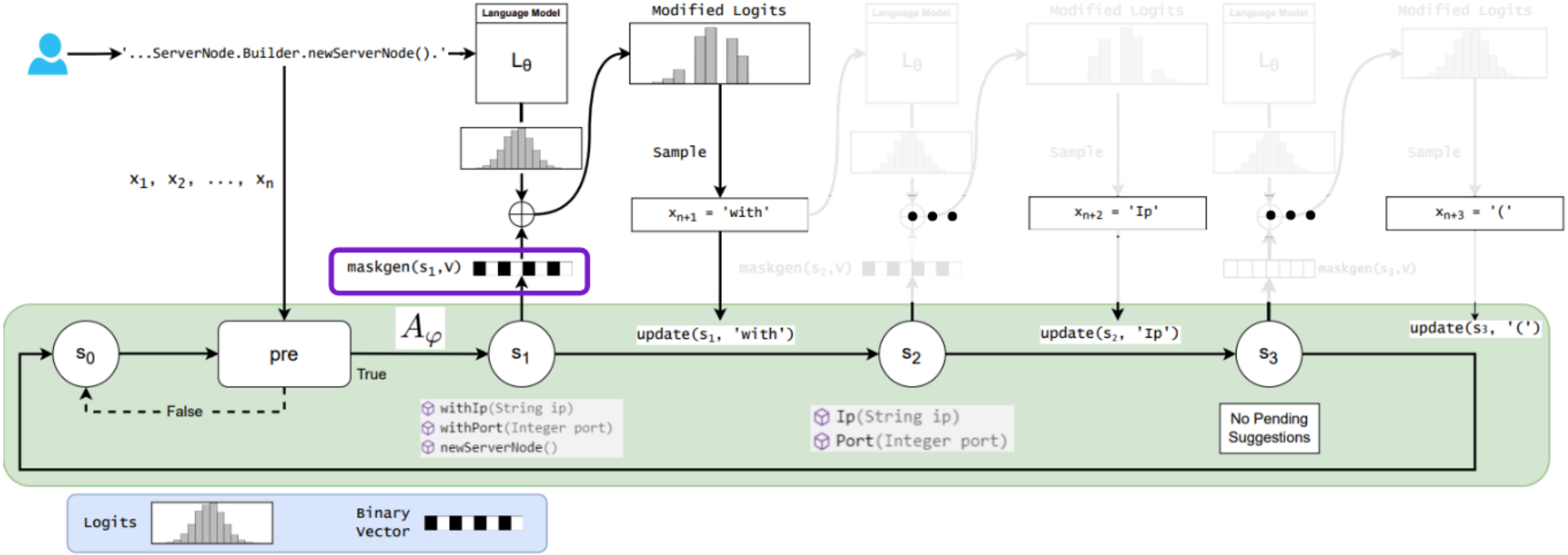



$A_\varphi$ Partial static analysis that derives constraints on the subsequent code at trigger location, such that the monitored property continues to be satisfied, for example, type-consistent identifier names

# Working of Monitor Guided Decoding

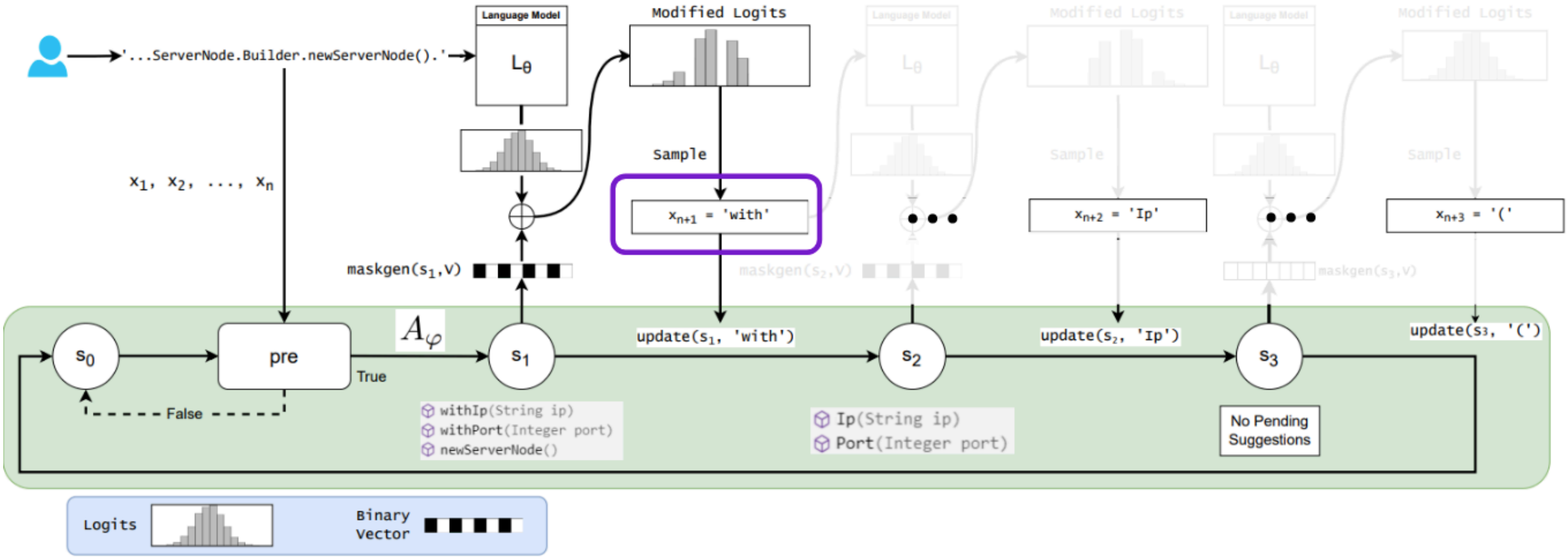


# Working of Monitor Guided Decoding

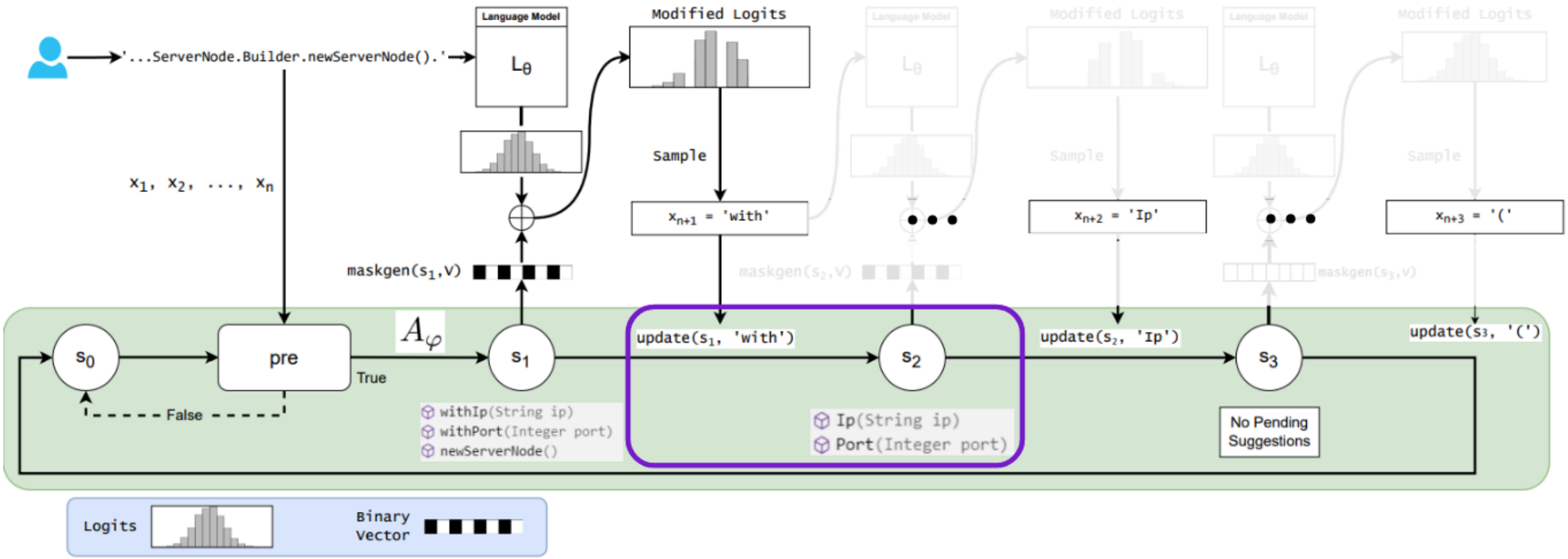



**maskgen** Identifies LM vocabulary tokens consistent with the current state of monitor. For example, selects tokens that are either prefix of any string in the current state, or of the form $w \cdot E \cdot \Sigma^*$, where w is a member of current state, E is a special set of non-identifier characters.

# Working of Monitor Guided Decoding



# Working of Monitor Guided Decoding




`update` Takes the current state, and decoded token as input, producing the next state consisting of updated constraints in light of the new token, or transitions back to the initial state, $s_0$

# Formalizing Monitor Guided Decoding

A Monitor $M_\varphi$ is a 6-tuple $(A_\varphi, s_0, S, \texttt{pre}, \texttt{update}, \texttt{maskgen})$

$$(L_\theta || M_\varphi)(x_{n+1}|x_1,\ldots,x_n;C,p,s) = \begin{cases} \texttt{softmax}(\ell)[X_{n+1}] & \text{if } s = s_0 \text{ is the wait state} \\ \texttt{softmax}(\ell \oplus m)[X_{n+1}] & \text{otherwise} \end{cases} \tag{1}$$

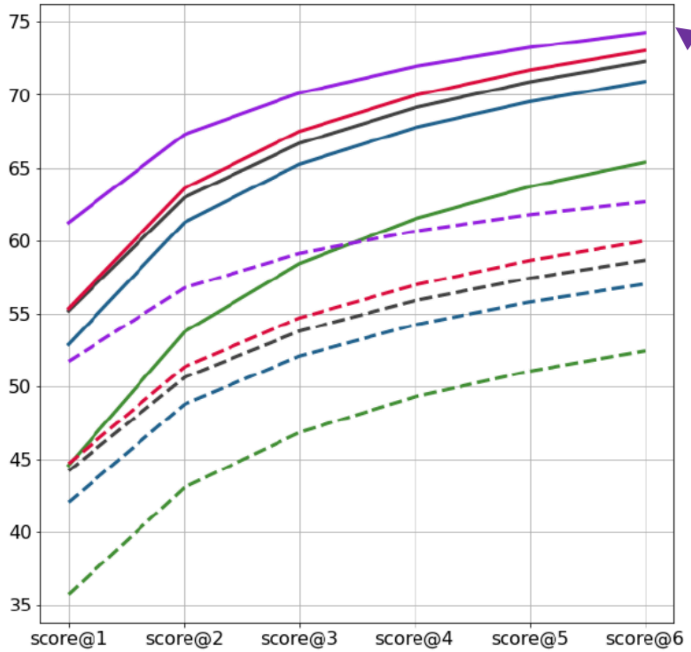
$$\ell = L_\theta(\,\cdot\,|x_1,\ldots,x_n;p) \tag{2}$$

$$m = \texttt{maskgen}(s, V) \tag{3}$$

$$s' = \begin{cases} A_\varphi(x_1,\ldots,x_n;C) & \text{if } s = s_0 \wedge \texttt{pre}(s;x_1,\ldots,x_n) \\ \texttt{update}(s, x_{n+1}) & \text{otherwise} \end{cases} \tag{4}$$

| | |
|---|---|
| pre | Precondition check – determines when to trigger the static analysis. |
| $A_\varphi$ | Partial static analysis that derives constraints on the subsequent code at trigger location, such that the monitored property continues to be satisfied, for example, type-consistent identifier names |
| $s_0$ $s_1$ $s_2$ … | $s_0$ is the default state in which all vocabulary tokens are valid. All the other states represent constraints to be applied for the next token. |
| `maskgen` | Identifies LM vocabulary tokens consistent with the current state of monitor. For example, selects tokens that are either prefix of any string in the current state, or of the form $w \cdot E \cdot \Sigma^*$, where w is a member of current state, E is a special set of non-identifier characters. |
| `update` | Takes the current state, and decoded token as input, producing the next state consisting of updated constraints in light of the new token, or transitions back to the initial state, $s_0$ |

# Monitor-guided Decoding




compilation rate

text-davinci-003

code-gen 350M

[Agrawal et al: *Monitor-guided decoding of code LMs with static analysis of repository context.* NeurIPS'23]

Thanks to monitor guidance,
a model with 1000x fewer parameters
can generate better code than GPT3!

# Techniques

Accuracy

- Constrained Decoding
- Fine Tuning

Validation

- Self-consistency
- User interaction

High-level DSL

# Self-Play

[Haluptzok et al: *Language models can teach themselves to program better.* ICLR'23]

AlphaZero got better at Go through self-play;
can we do this for code?

*Idea:* use LLM to generate *programming puzzles* and solutions to those puzzles

```
def f(c: int):
    return c + 50000 == 174653

def g():
    return 174653 - 50000

assert f(g())
```

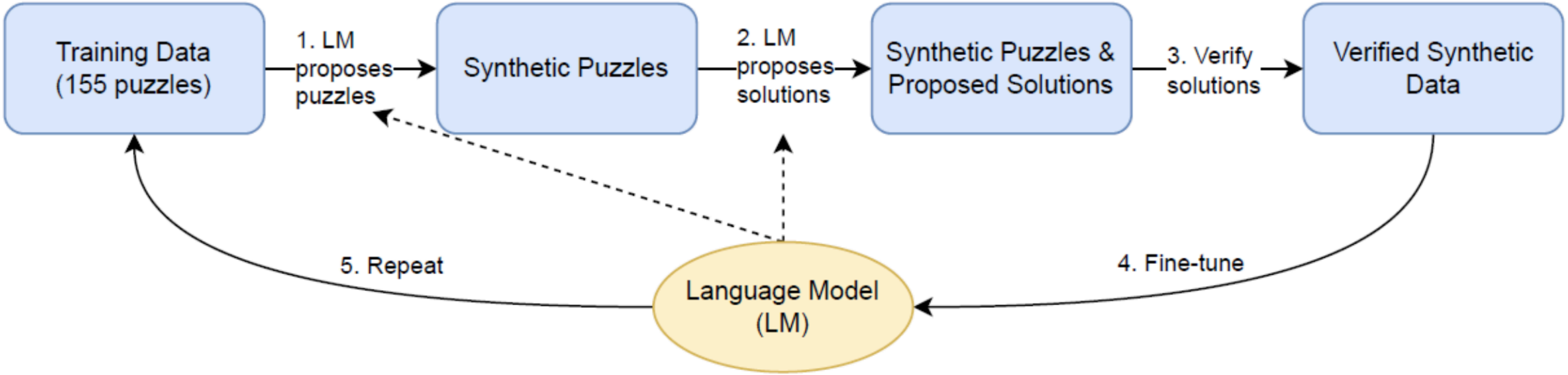
```
def f(x: str, chars=['Hello', 'there', 'you!'], n=4600):
    return x == x[::-1] and all([x.count(c) == n for c in chars])

def g(chars=['Hello', 'there', 'you!'], n=4600):
    s = "".join([c*n for c in chars])
    return s + s[::-1]

assert f(g())
```

# Self-Play

[Haluptzok et al: *Language models can teach themselves to program better.* ICLR'23]



# Techniques

Accuracy

- Constrained Decoding
- Fine Tuning

Validation

- Self-consistency
- User interaction

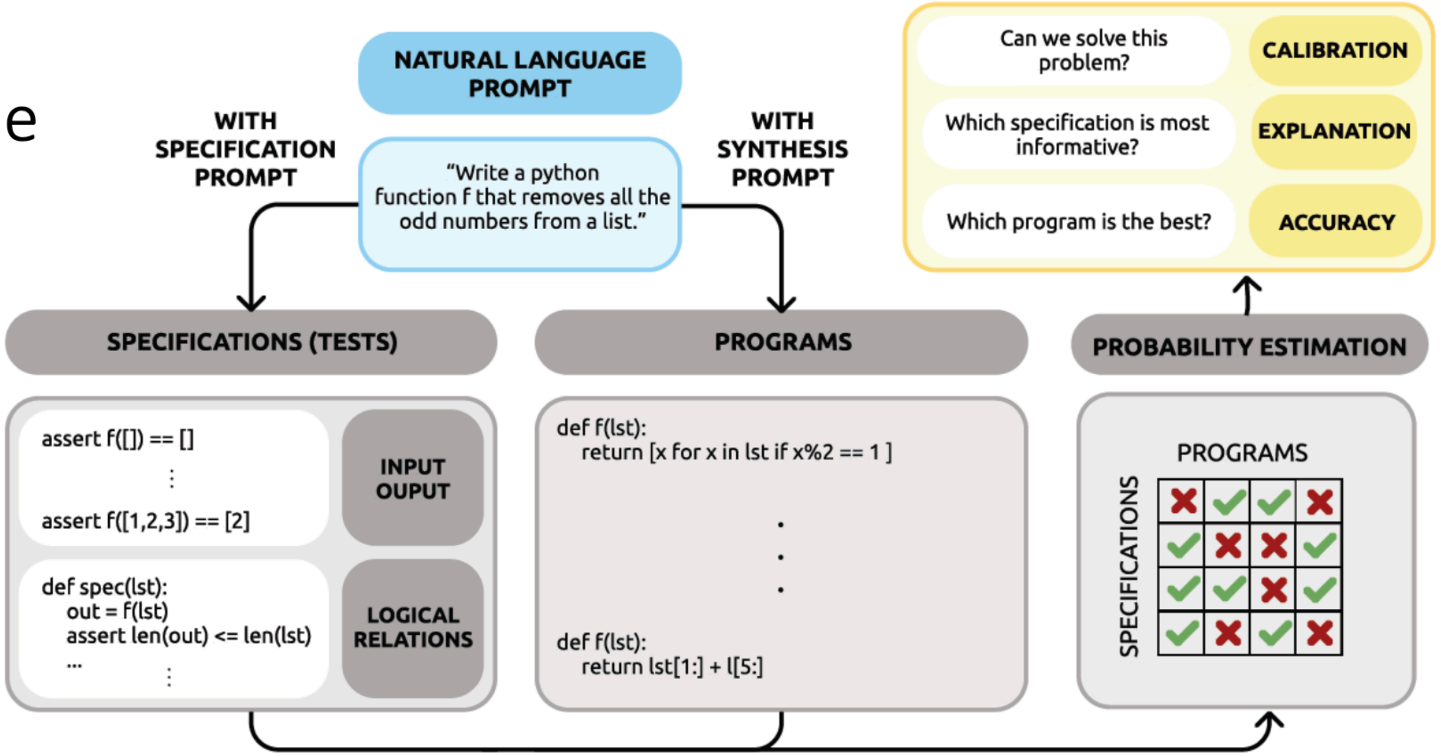
High-level DSL

# Speculyzer

[Li, Key, Ellis: *Towards trustworthy neural program synthesis.* 2023]

*Goal*: Increase trustworthiness of NL->code

*Idea:* generate *tests* alongside programs

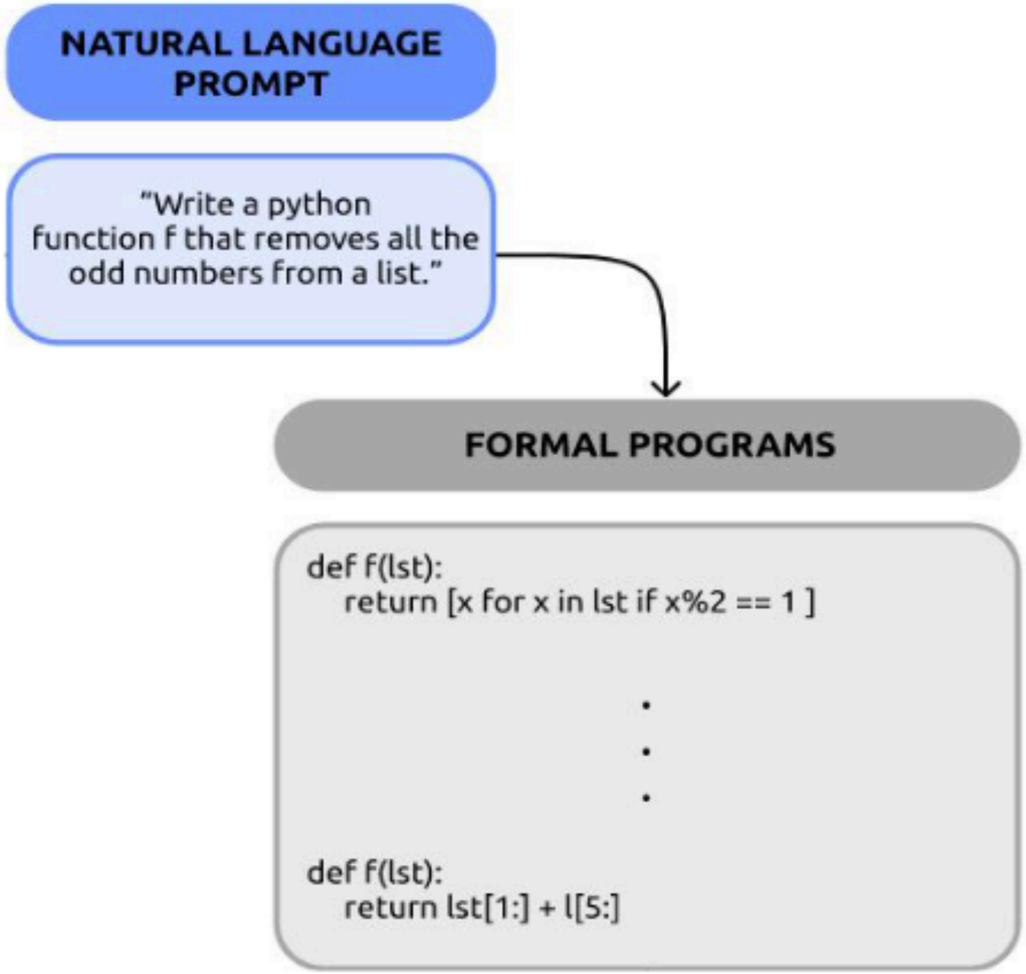


# Speculyzer

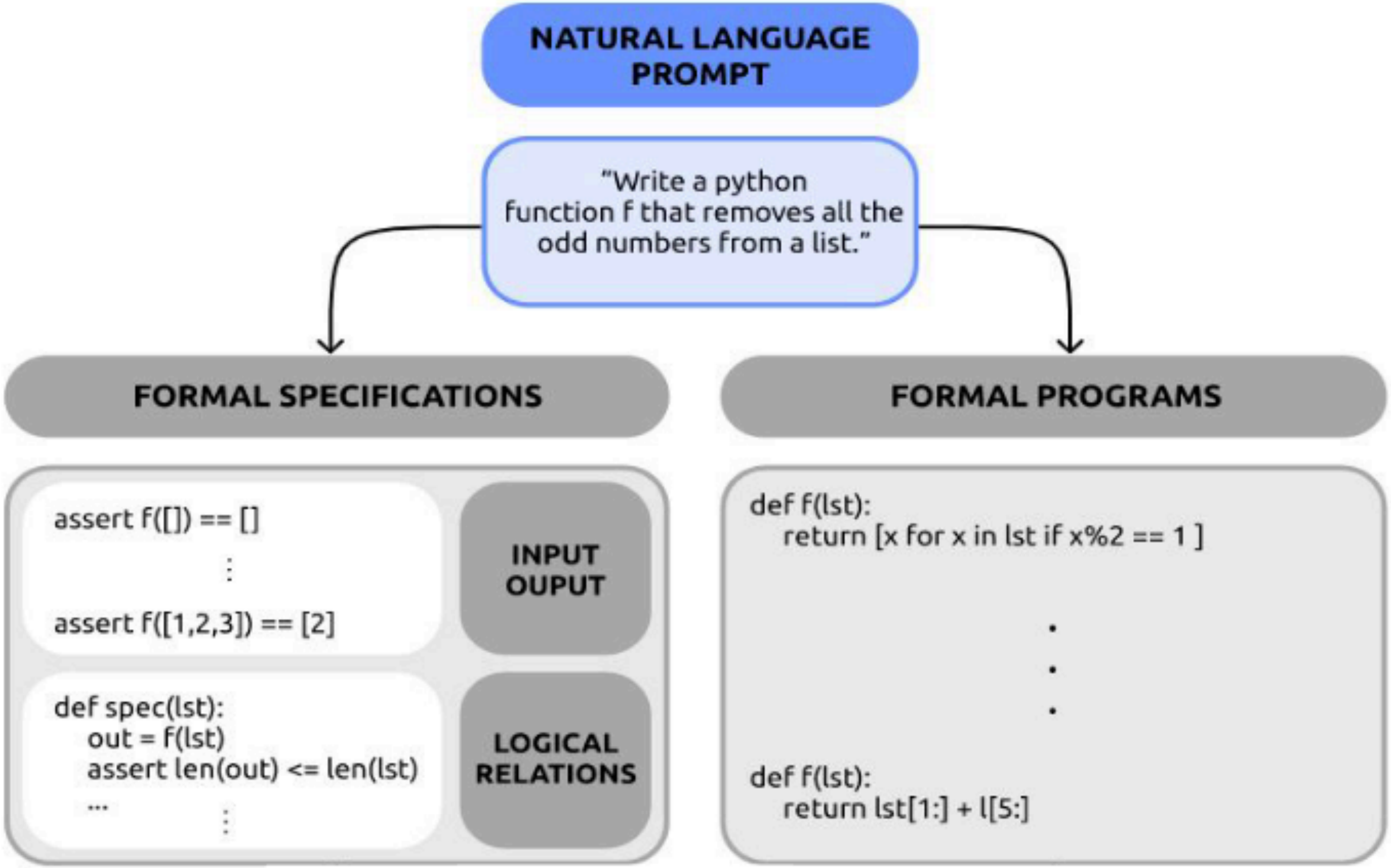
**NATURAL LANGUAGE PROMPT**

"Write a python function f that removes all the odd numbers from a list."

# Speculyzer

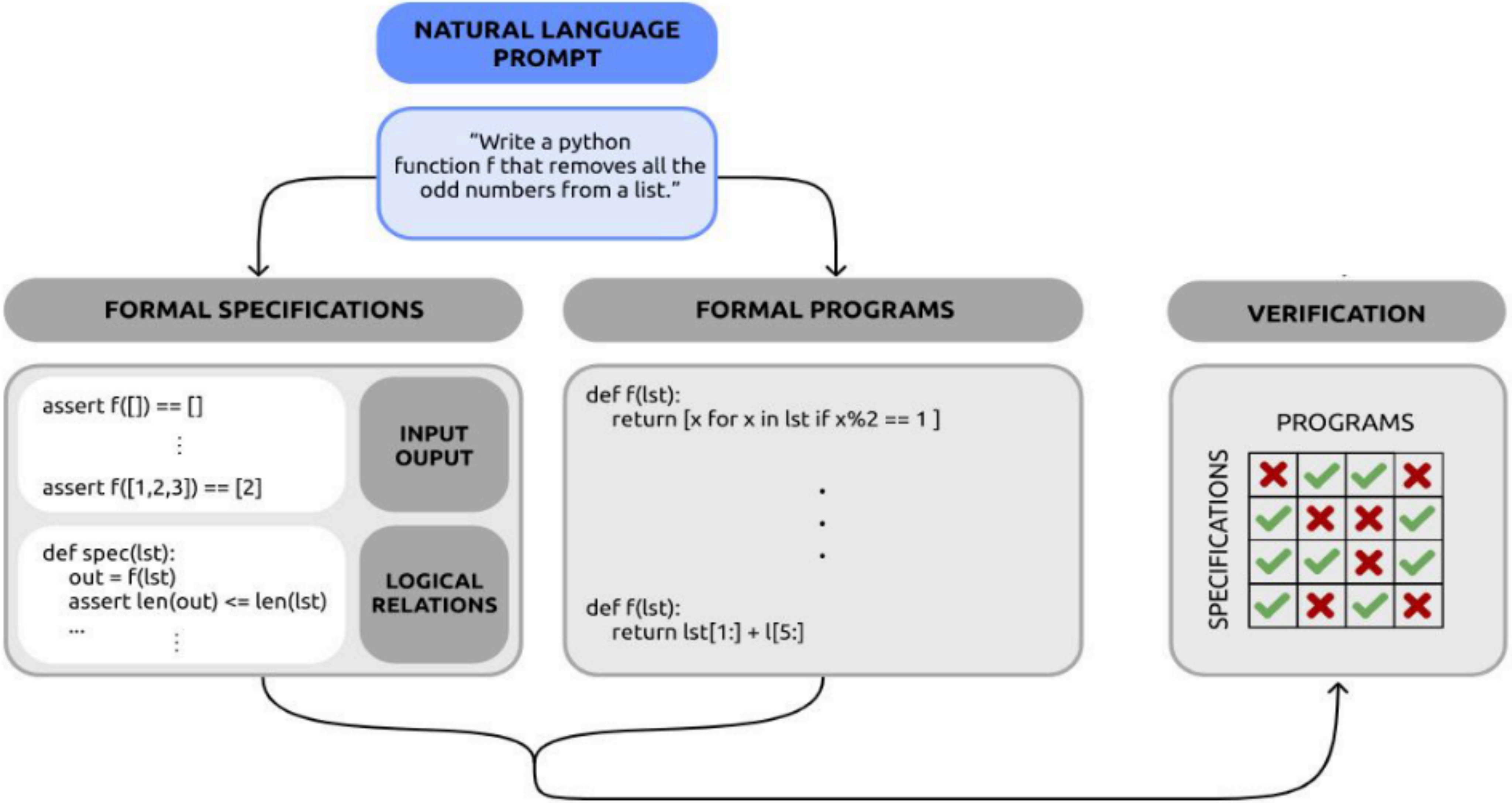

NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
FORMAL PROGRAMS
def f(lst):
return [x for x in lst if x%2 == 1 ]
.
.
.
def f(lst):
return lst[1:] + l[5:]

# Speculyzer



NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
FORMAL SPECIFICATIONS
assert f([]) == []
⋮
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
⋮
LOGICAL RELATIONS
FORMAL PROGRAMS
def f(lst):
return [x for x in lst if x%2 == 1 ]
def f(lst):
return lst[1:] + l[5:]

# Speculyzer



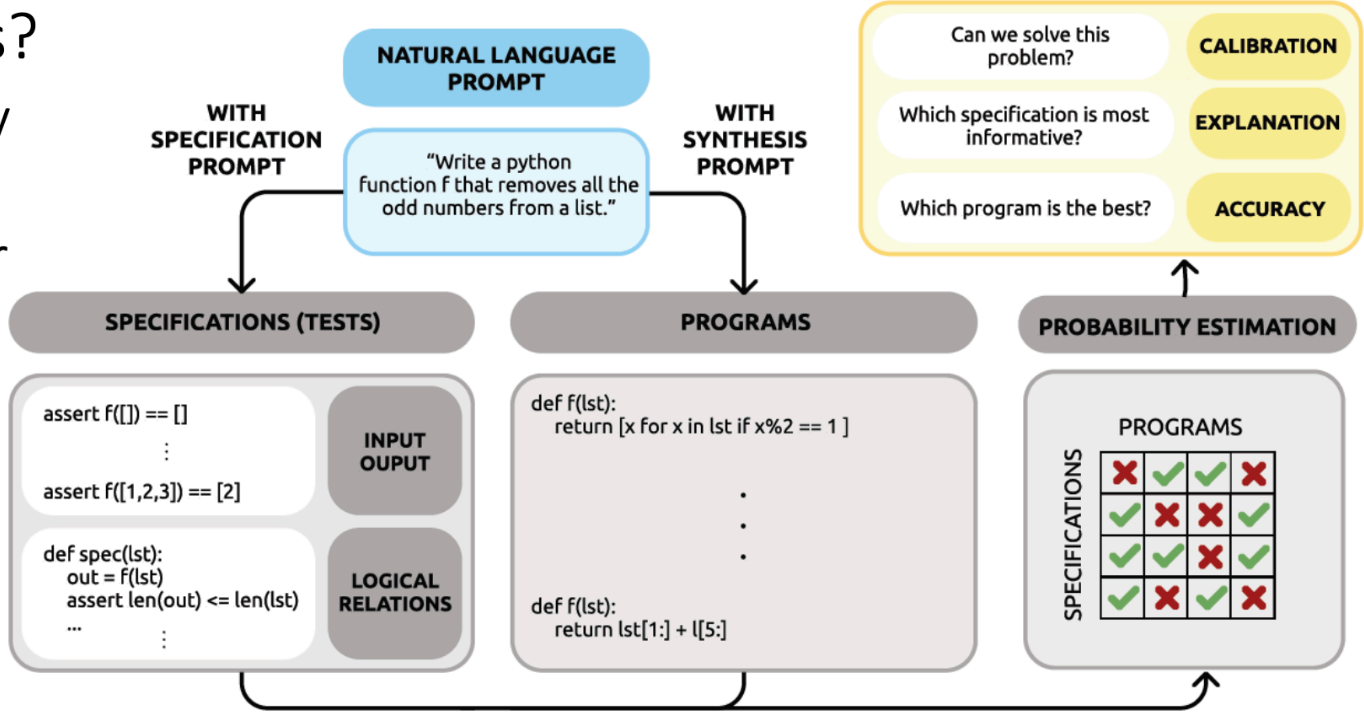
NATURAL LANGUAGE PROMPT
"Write a python function f that removes all the odd numbers from a list."
FORMAL SPECIFICATIONS
assert f([]) == []
assert f([1,2,3]) == [2]
INPUT OUPUT
def spec(lst):
out = f(lst)
assert len(out) <= len(lst)
...
LOGICAL RELATIONS
FORMAL PROGRAMS
def f(lst):
return [x for x in lst if x%2 == 1 ]
def f(lst):
return lst[1:] + l[5:]
VERIFICATION
PROGRAMS
SPECIFICATIONS

# Speculyzer

[Li, Key, Ellis: *Towards trustworthy neural program synthesis.* 2023]

What can we do with the tests?

- rank programs based how many tests they pass
- cluster programs based on their behavior on test inputs
- train a classifier to predict if the model knows the solution
- pick the most selective tests to show to the user



# Speculyzer

**PROGRAM**

```
def derivative(xs: list):
    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
     Return derivative of this polynomial in the same form.
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
    return [x * i for i, x in enumerate(xs) if i != 0]
```

**TOP LOGICAL RELATION**

```
def test_derivative(xs: list):
    """ Given an input `xs`, test whether the function `derivative`
is implemented correctly.
    """
    ys = derivative(xs)
    assert len(ys) == len(xs) - 1
    for i in range(len(ys)):
        assert ys[i] == xs[i+1] * (i + 1)

# run `test_derivative` on a new testcase
test_derivative([3, 1, 2, 4, 5])
```

**RANDOM LOGICAL RELATION**

```
def test_derivative(xs):
    """ Test function derivative().
    """
    # TODO
    pass

# run `test_derivative` on a new testcase
test_derivative([2, 3, 4, 10, -12])
```

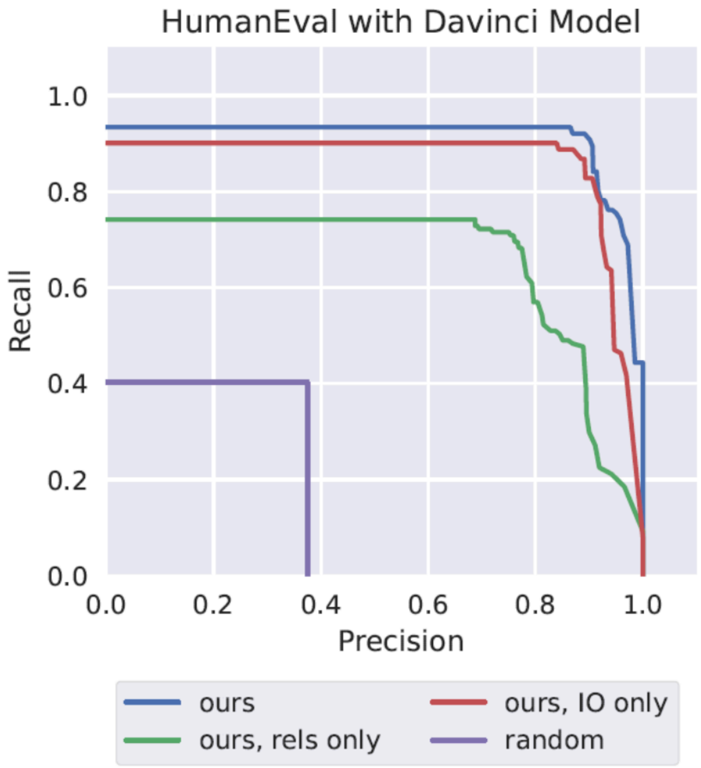
[Li, Key, Ellis: *Towards trustworthy neural program synthesis.* 2023]

Picking the most selective test to show to the user

# Speculyzer: results

Can achieve *zero error rate* on human eval in exchange for dropping recall from 93% to 44%!

[Li, Key, Ellis: *Towards trustworthy neural program synthesis.* 2023]



# Techniques

Accuracy

- Constrained Decoding
- Fine Tuning

Validation

- Self-consistency
- User interaction

High-level DSL

# The validation challenge

*"In the context of Copilot, there is a shift from writing code to understanding code"*

Taking Flight with Copilot, ACM Queue, Dec 22

validation is hard

- [Vaithilingam et al] observed 8 cases of over-reliance: bugs due to skipped validation

validation is a bottleneck

- single most prevalent activity according to [Mozannar et al]

prevalence of a validation strategy depends on its cost [Liang et al]

to help with validation, we need to lower its cost

# LEAP

[Ferdowsi et al: *Validating AI-Generated Code with Live Programming*. CHI'24]

lowers the cost of validation by execution
using live programming

# Research questions

how does live programming affect…

over- / under-reliance on AI

validation strategies

cognitive load

# Tasks

API-heavy

multiple correct suggestions

**pandas**

clean dataframe and compute stats using pandas

algorithmic

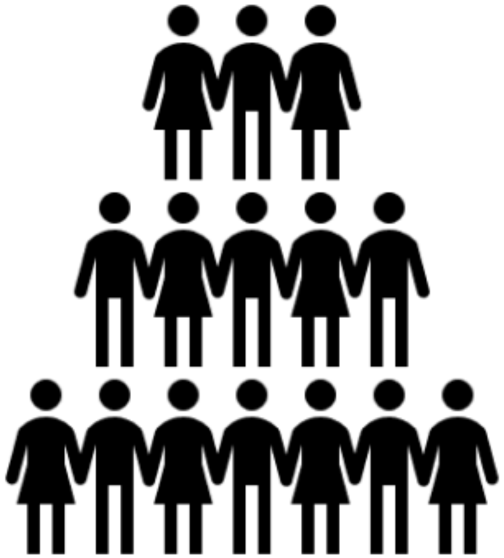
no correct suggestions

**bigrams**

find most frequent bigram in a string

fixed prompt

# Participants

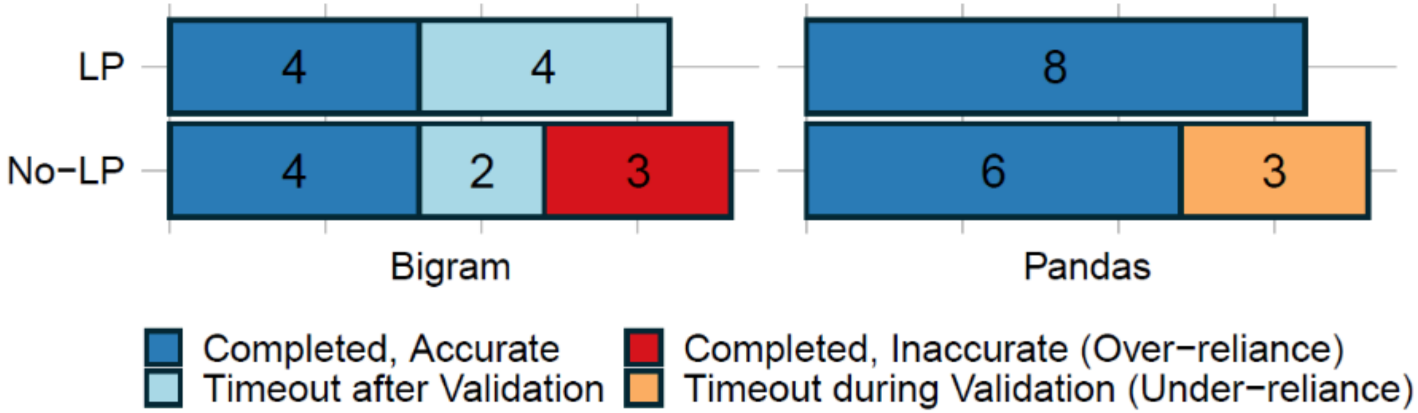


n = 17

occupation:
15 academia / 2 industry

Python usage:
2 occasionally /
8 regularly /
7 almost every day

# RQ1: over-/under-reliance




6 no-PB vs 0 PB participants **mid-judged** correctness of their solution

by lowering the cost of validation,
leap reduces over-/under-reliance on AI

# RQ1: over-/under-reliance

"it was **easy to understand** the behavior of a code suggestion because the little boxes on the side allowed for you to preview the results." (P3)

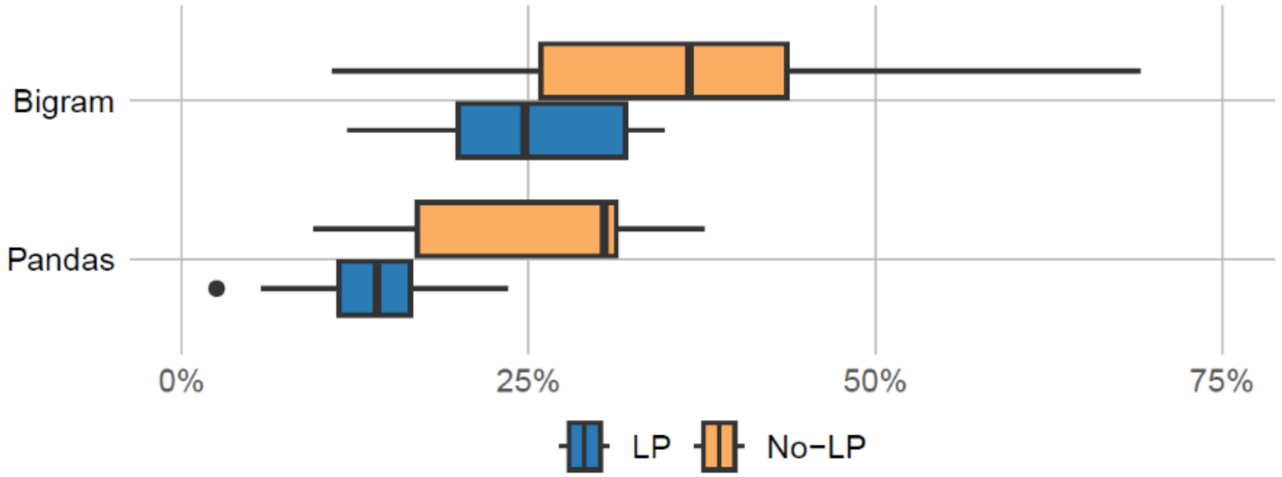
"it **saved me the effort** of writing multiple print statements." (P1)

6 no-PB vs 0 PB participants **mid-judged** correctness of their solution

by lowering the cost of validation,
leap reduces over-/under-reliance on AI

# RQ2: validation strategies

percentage of time spent in Suggestion Panel




“I didn’t look too closely in the actual code,
I was *just looking at the runtime values* on the side.” (P1)

leap participants spent less time reading code

# Techniques

Accuracy

- Constrained Decoding
- Fine Tuning

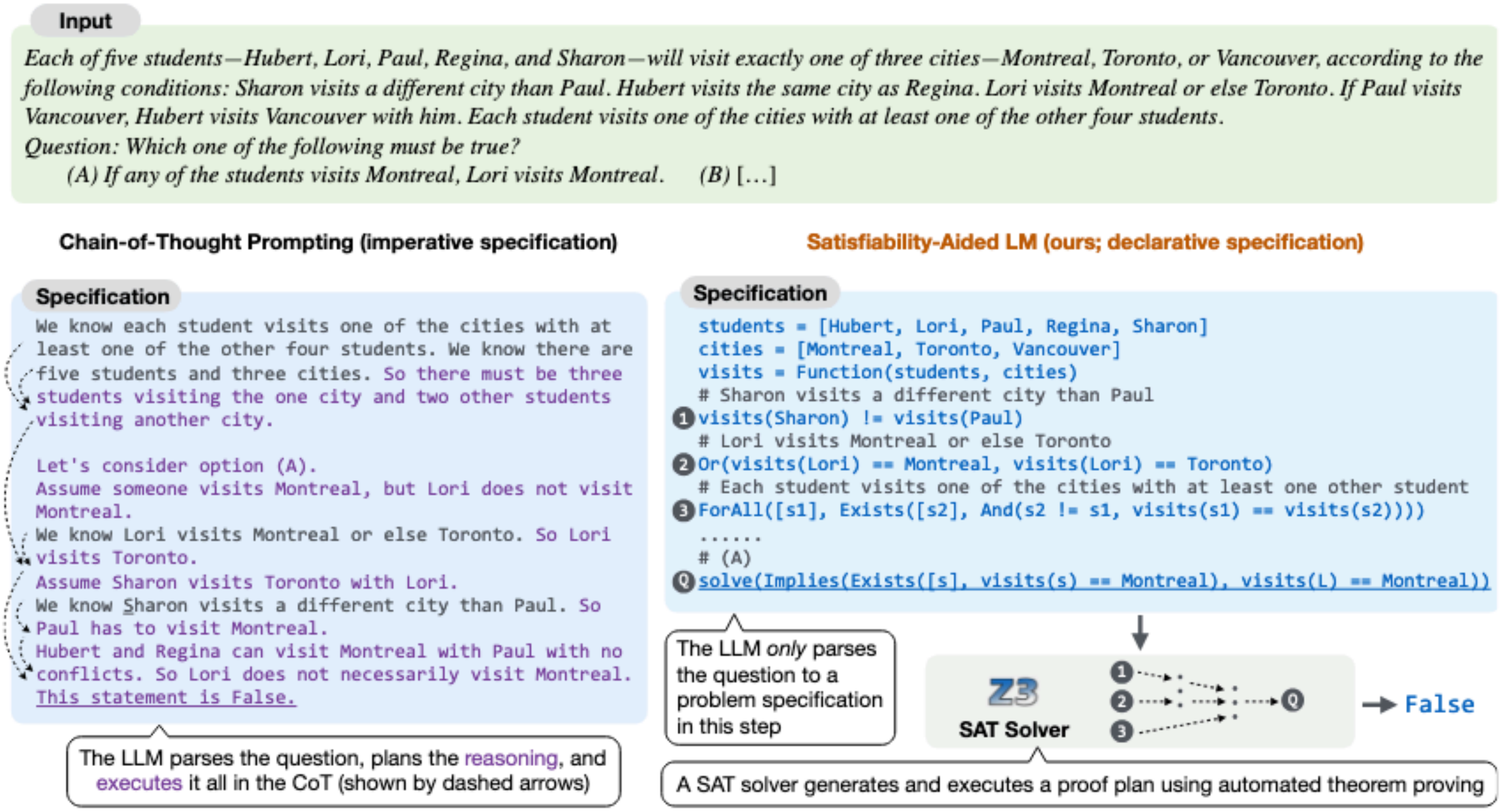
Validation

- Self-consistency
- User interaction

High-level DSL

# SatLM

Xi Ye, Qiaochu Chen, Isil Dillig, Greg Durrett: [SatLM: Satisfiability-Aided Language Models Using Declarative Promptin](#)
NeurIPS'23



# SatLM: Potential Improvements

Run multiple times and

- ignore attempts that don’t parse or produce AMBIG/UNSAT
- even better: check answers for consistency

Run in a loop, providing feedback to the LLM

- if AMBIG, tell the LLM to strengthen the constraints
- if UNSAT, get UNSAT core and tell the LLM to weaken one of those

Combine individual constraints from different solutions

- maybe perform lattice search until we get a SAT, unambiguous set

# Logistics

- Final Exam : November 27 , Timings?
- Project Presentations November 29th (4:30 - 6 pm)
- Project Report Submission Dec: 2nd
- Exam marks : Tuesday
- Please see your ERP for the marks for the paper reading.
- Syllabus:
  - Starting Sketch and CS
  - Minus Hoare Logic