Microsoft Microsoft Research

Monitor-Guided Decoding of Code LMs with Static Analysis of Repository Context

Lakshya A Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K. Lahiri, Sriram Rajamani

Code and Data
aka.ms/monitors4codegen

Link to Paper
aka.ms/mgd_paper

NEURAL INFORMATION PROCESSING SYSTEMS

Presenting at NeurIPS 2023
Thirty-seventh Conference on Neural Information Processing Systems, New Orleans

Introduction

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

Introduction

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

Introduction

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

```
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()
            .
}
```

Introduction

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

```
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**

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

Introduction

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.

Method to be completed

```
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**

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

Monitor Guided Decoding

Method to be completed

```
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**

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

✓ **SantaCoder with *monitor guided decoding***

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

Monitor Guided Decoding

Method to be completed

```
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**

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

✓ **SantaCoder with *monitor guided decoding***

```
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.

Monitor Guided Decoding

Method to be completed

```
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**

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

✓ **SantaCoder with *monitor guided decoding***

```
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.

Monitor Guided Decoding

Method to be completed

```
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**

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

✓ **SantaCoder with *monitor guided decoding***

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

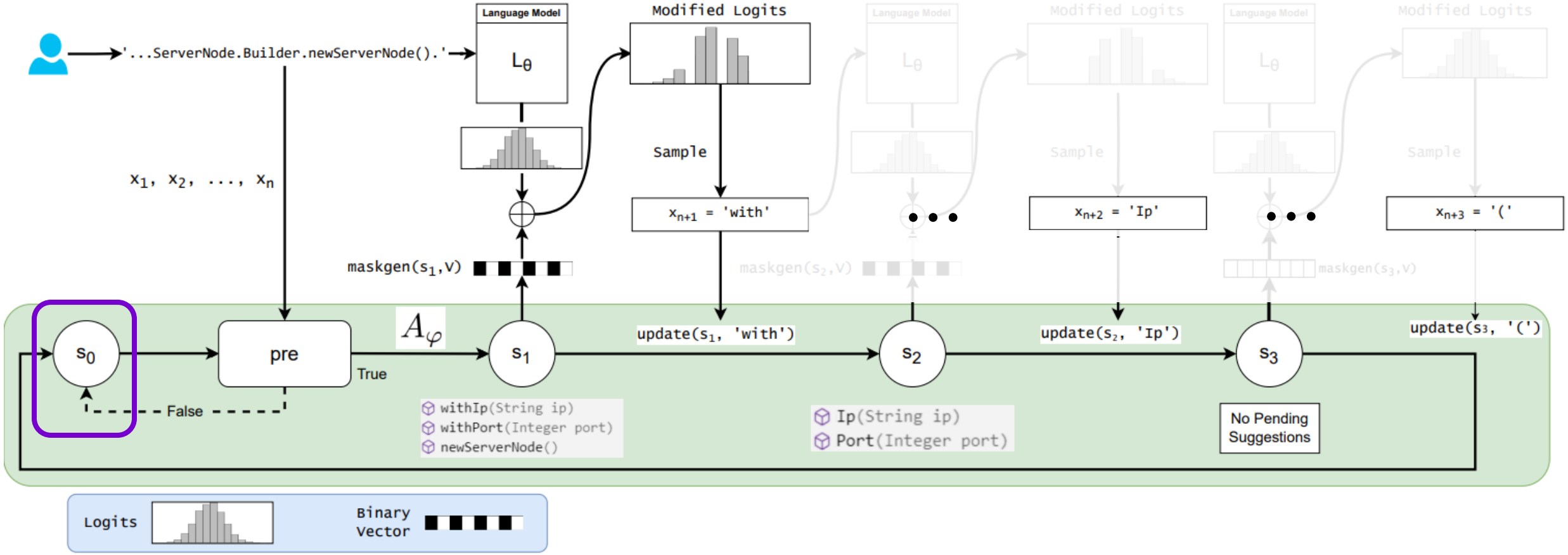
Intuition: IDEs assist human developers by providing global context information during code authoring. We extend this IDE assistance to LMs.

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

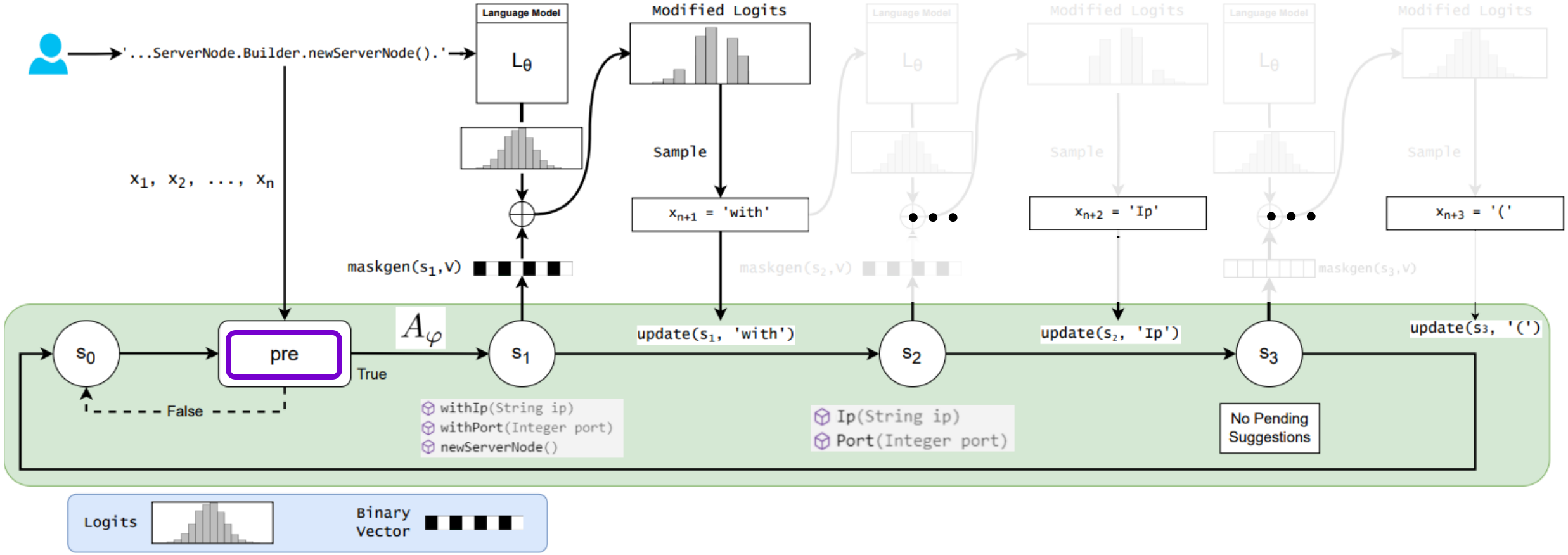
Working of Monitor Guided Decoding



s_0 s_1 s_2 …

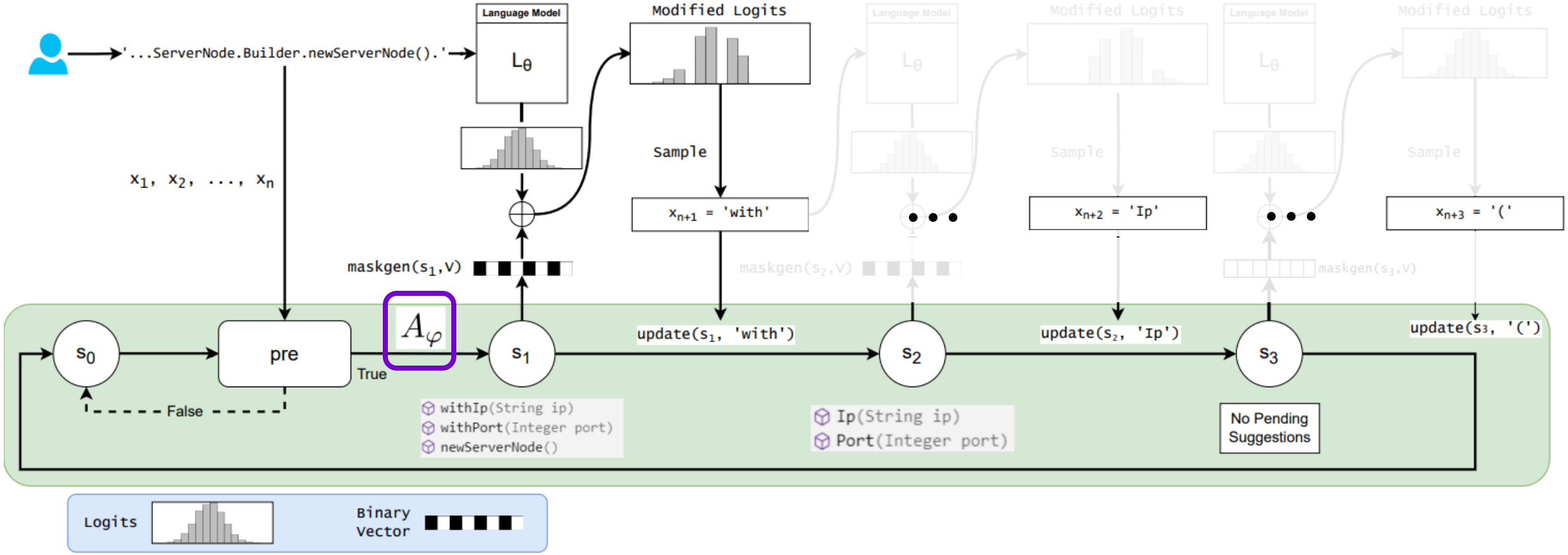
s_0 is the default state in which all vocabulary tokens are valid. All the other states represent constraints to be applied for the next token.

Working of Monitor Guided Decoding



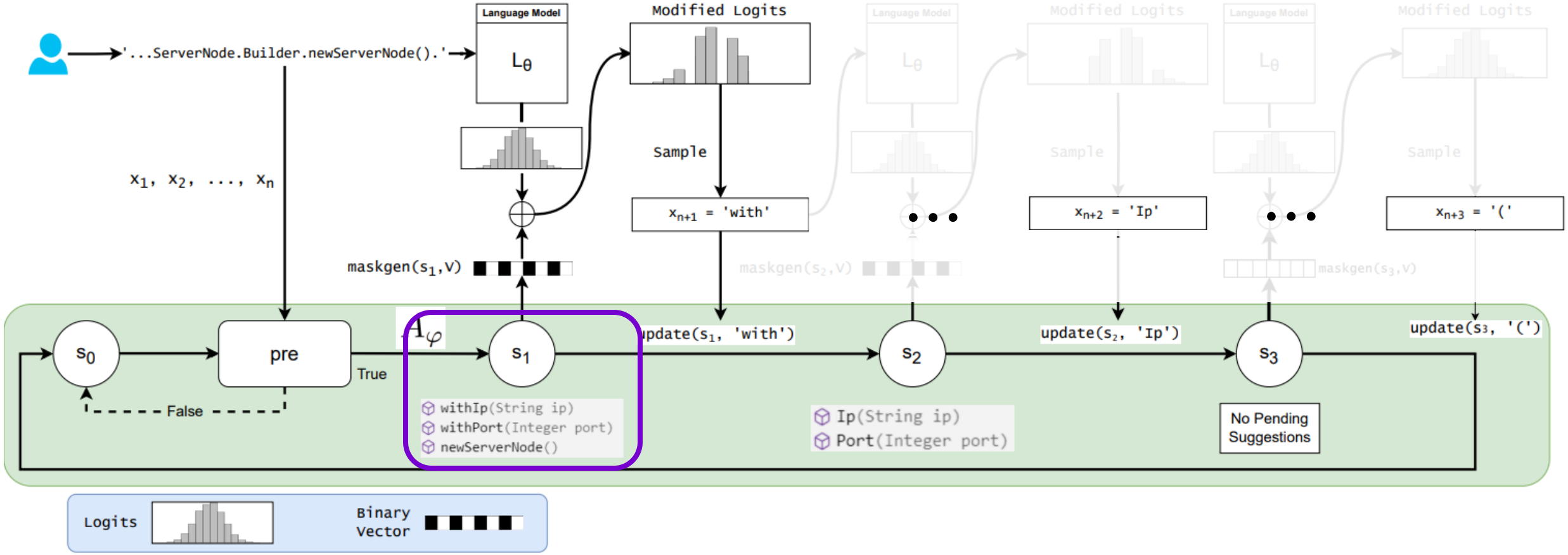
pre Precondition check – determines when to trigger the static analysis.

Working of Monitor Guided Decoding

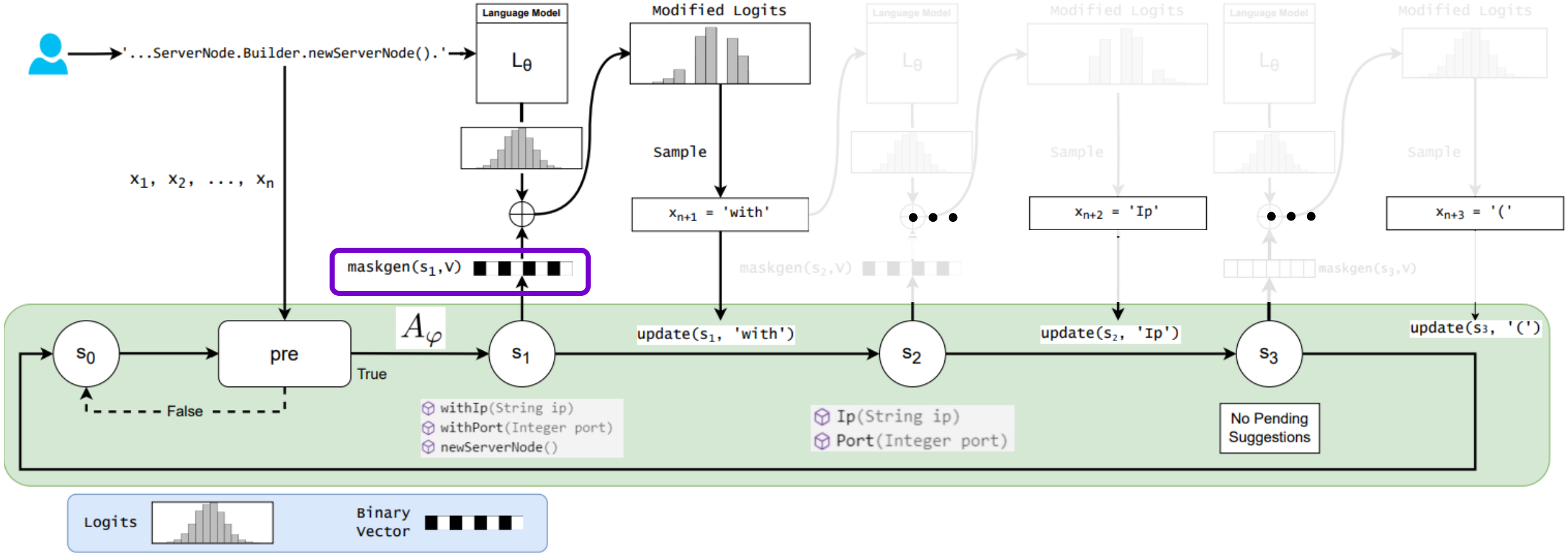


A_φ Partial static analysis that derives constraints on the subsequent code at trigger location, such that the monitored property continues to be satisfied, for example, type-consistent identifier names

Working of Monitor Guided Decoding

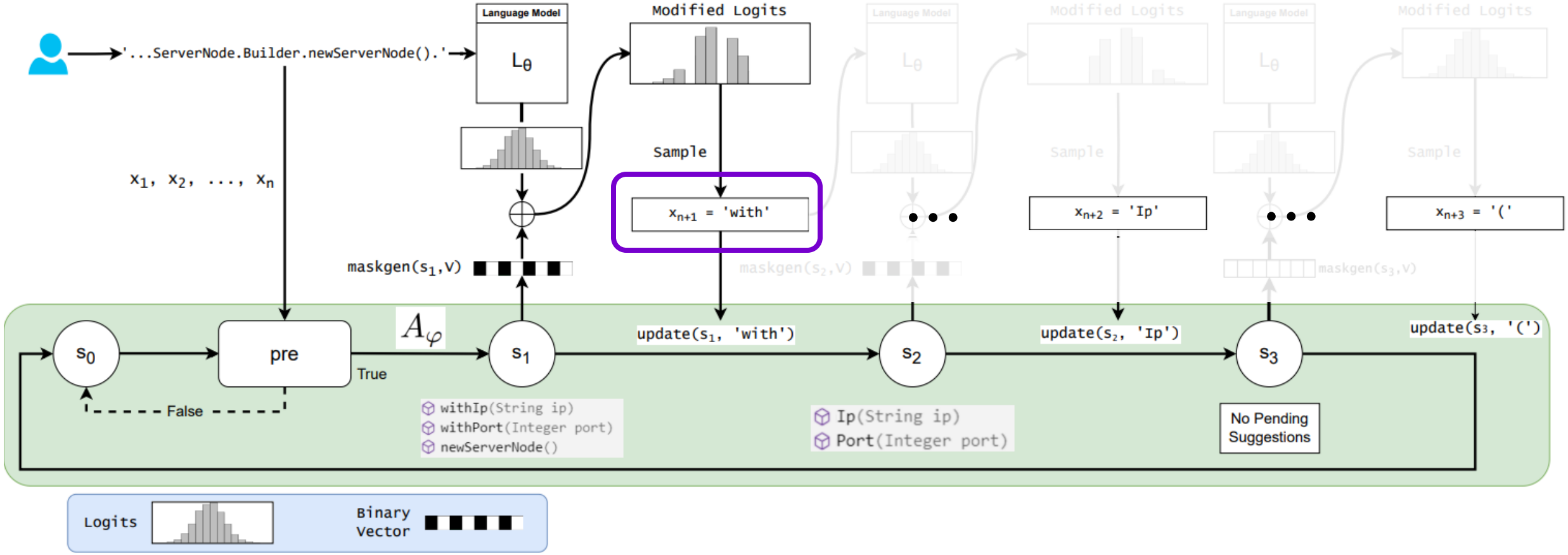


Working of Monitor Guided Decoding

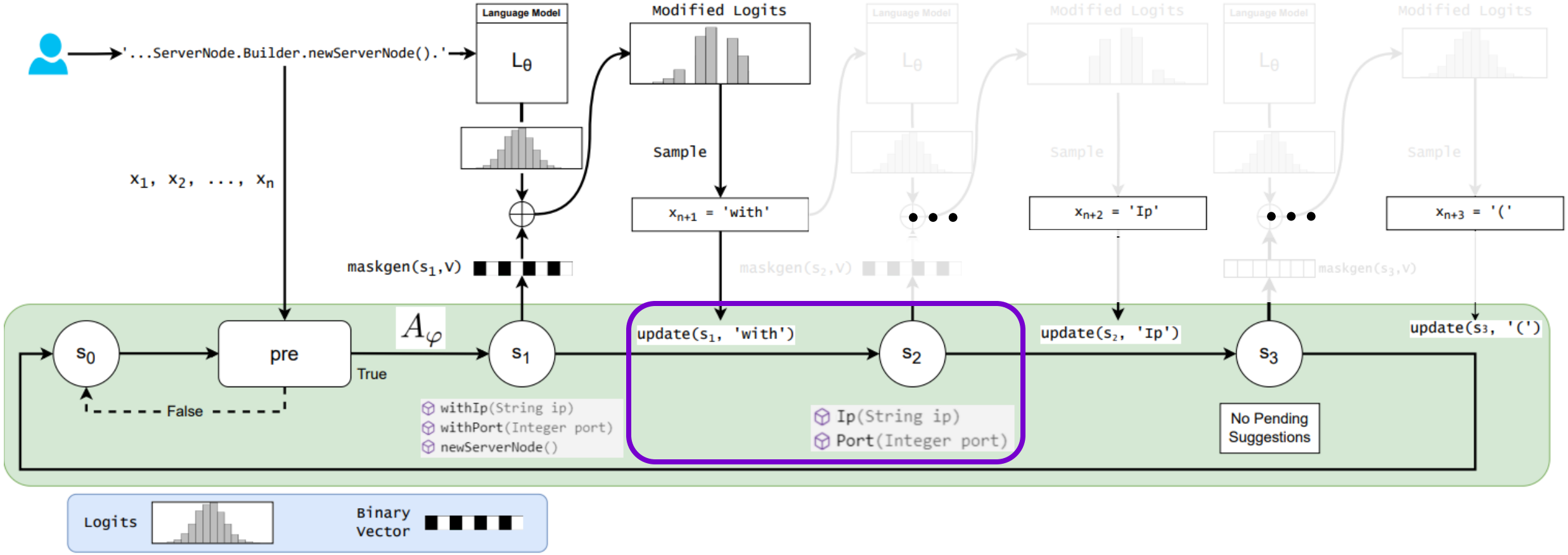


`maskgen` Identifies LM vocabulary tokens consistent with the current state of monitor. For example, selects tokens that are either prefix of any string in the current state, or of the form $w \cdot E \cdot \Sigma^*$, where w is a member of current state, E is a special set of non-identifier characters.

Working of Monitor Guided Decoding

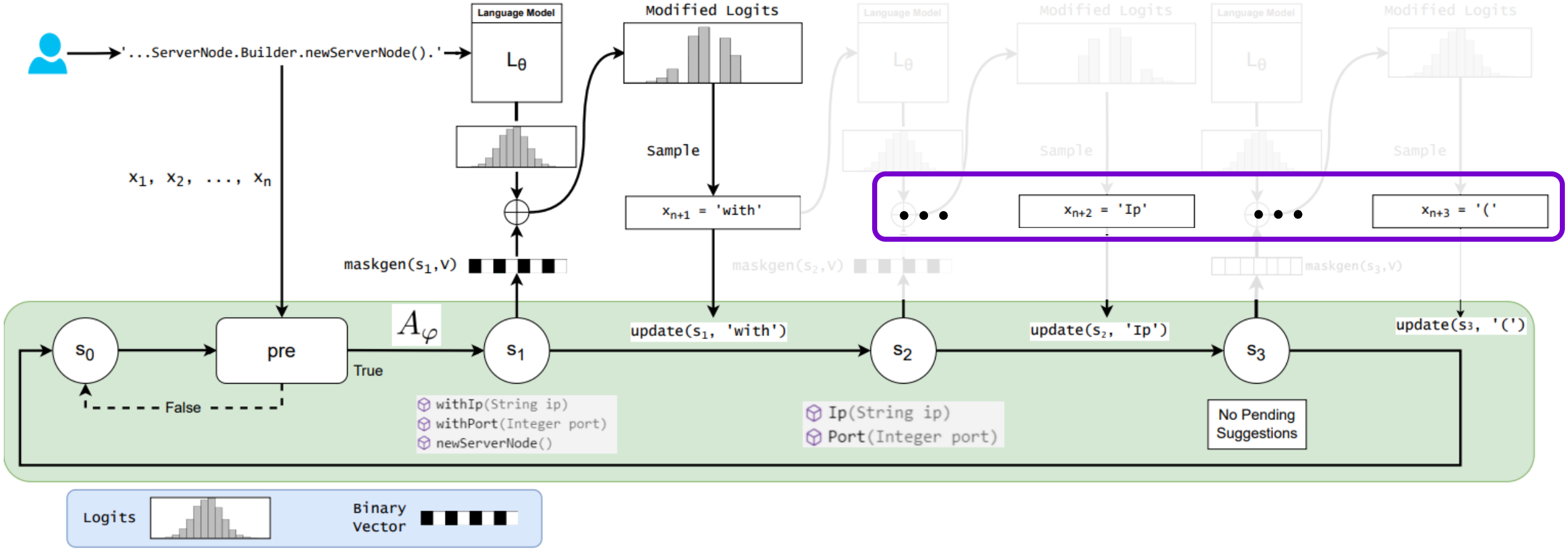


Working of Monitor Guided Decoding

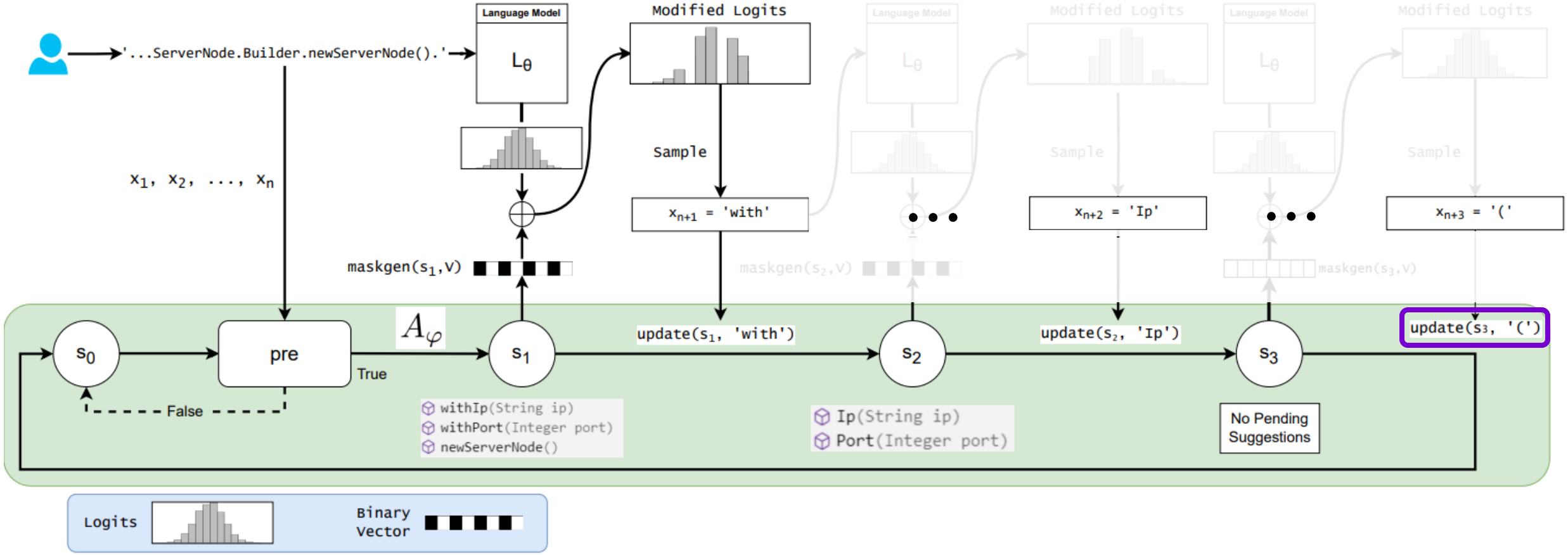


`update` Takes the current state, and decoded token as input, producing the next state consisting of updated constraints in light of the new token, or transitions back to the initial state, s_0

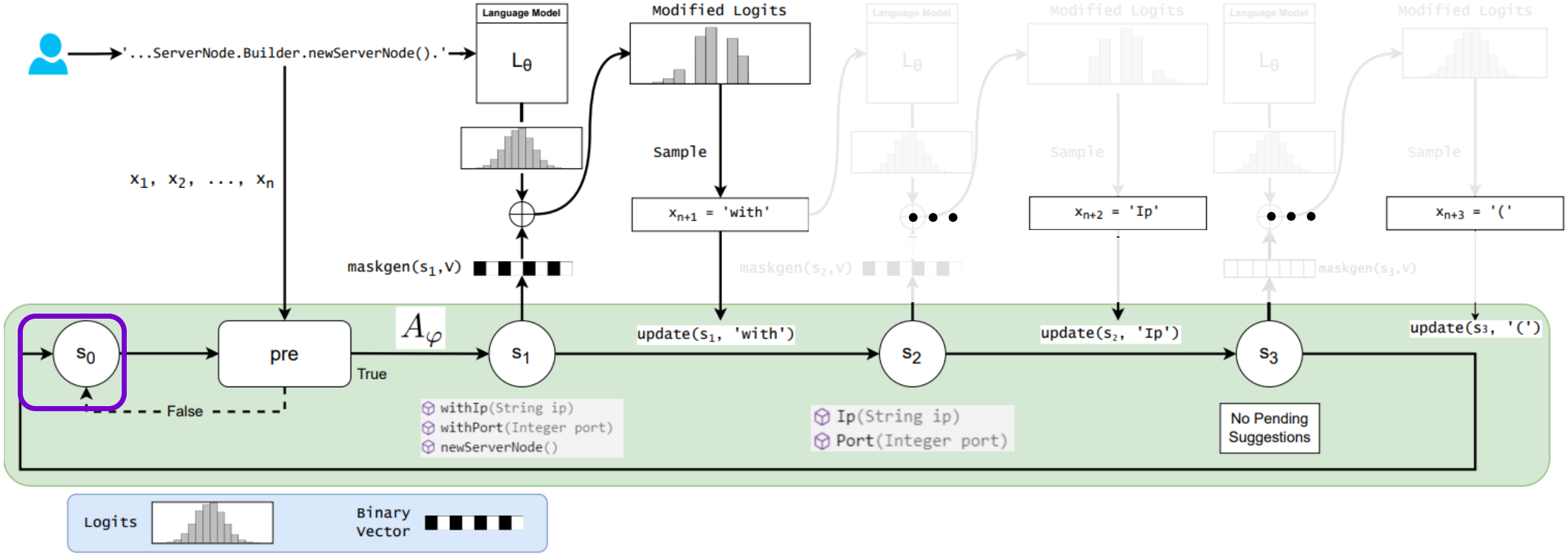
Working of Monitor Guided Decoding



Working of Monitor Guided Decoding



Working of Monitor Guided Decoding



Formalizing Monitor Guided Decoding

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

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

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

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

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

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

Evaluation & Results: Experimental Setup

PragmaticCode and DotPrompts: Java Evaluation Dataset

# of Repositories	100
# of Methods	1420
# of Testcases	10538

- Each testcase consists of a prompt up to a dereference point in a target method
- Task: method-completion utilizing repository-level context
- For evaluation, 6 generations are sampled for each testcase

Models

CG-350M	Salesforce CodeGen-350M-Multi
CG-2B	Salesforce CodeGen-2B-Multi
CG-6B	Salesforce CodeGen-6B-Multi
SC	BigCode SantaCoder-1.1B
TD-3	OpenAI text-davinci-003 (175B)

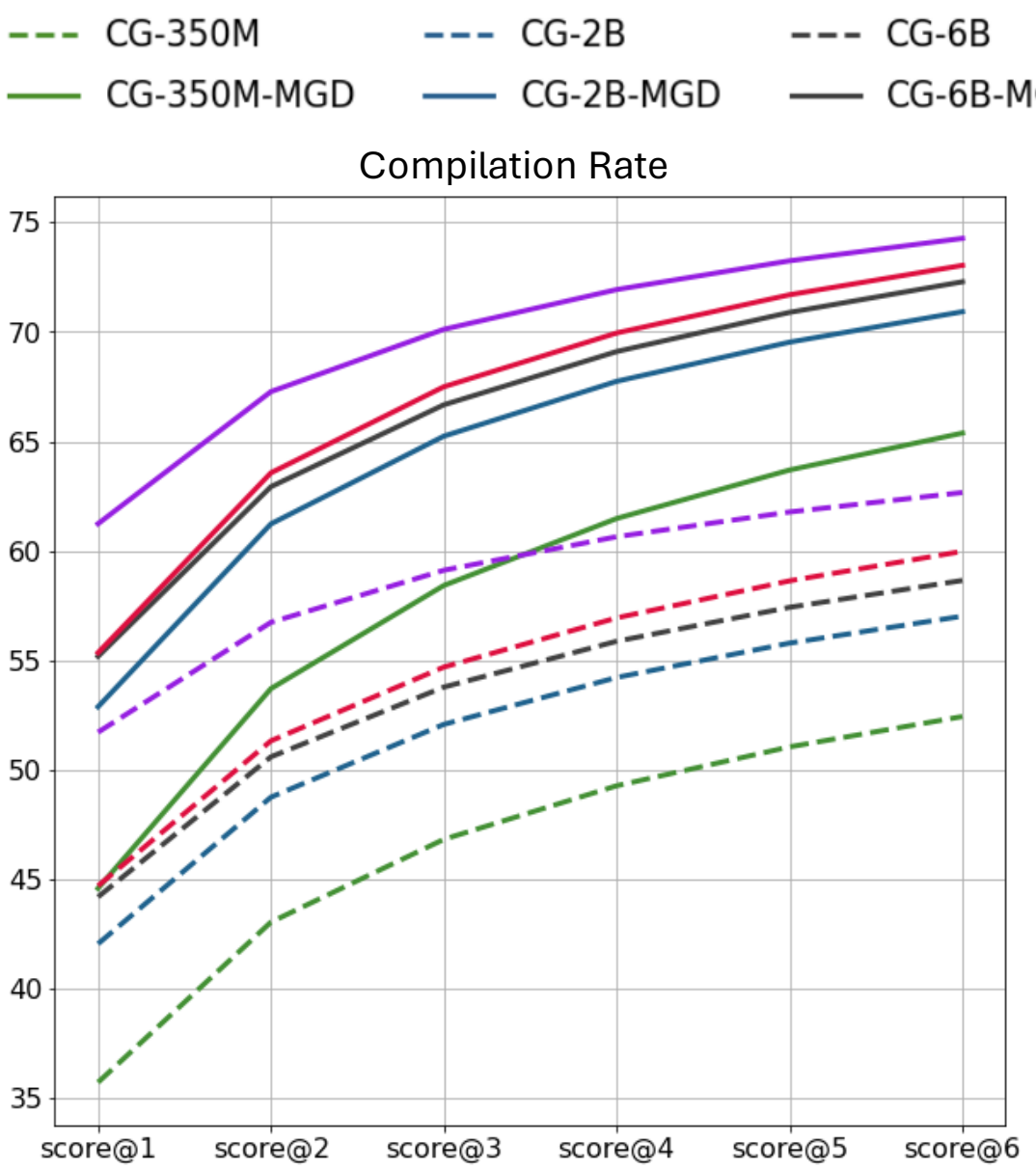
Prompting Baselines

Standard	Prompt consists of only the target method file content
classExprTypes	Including cross-file type information in prompt
RLPG	Including cross-file information by learning to predict from 60+rules
FIM	Use of the fill-in-the-middle capabilities of the model

Evaluation Metrics

CR	Compilation Rate: Fraction of testcases, for which generated code compiled successfully
NIM	Next Identifier Match: Fraction of testcases, for which generated next identifier is accurate
ISM	Identifier Sequence Match: Percent prefix of ordered identifiers in the ground truth matched by the generated code
PM	Prefix Match: Percent prefix of ground truth matched by generated code

Improvements in Compilation Rate

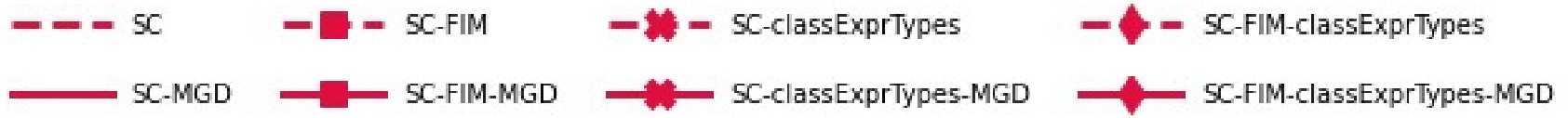


Every LM with MGD *(irrespective of model size and architecture)* **20-25% relative improvement in compilation rate**

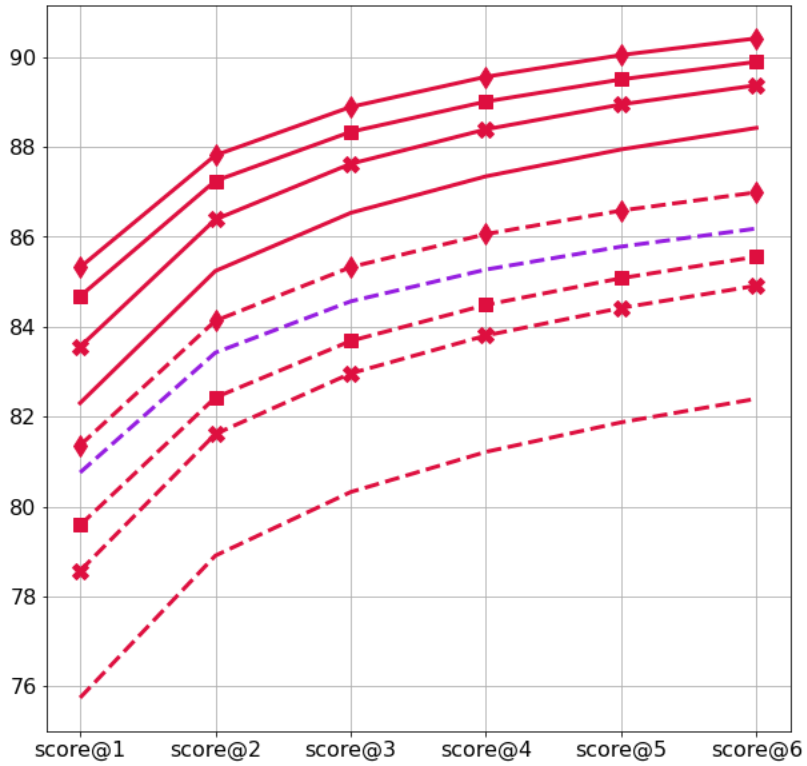
CodeGen-350M with MGD beats the 500x larger **text-davinci-003**

SantaCoder-1.1B with MGD improves over compilation rate of **text-davinci-003 (without MGD)** by a large relative margin of 16.5%

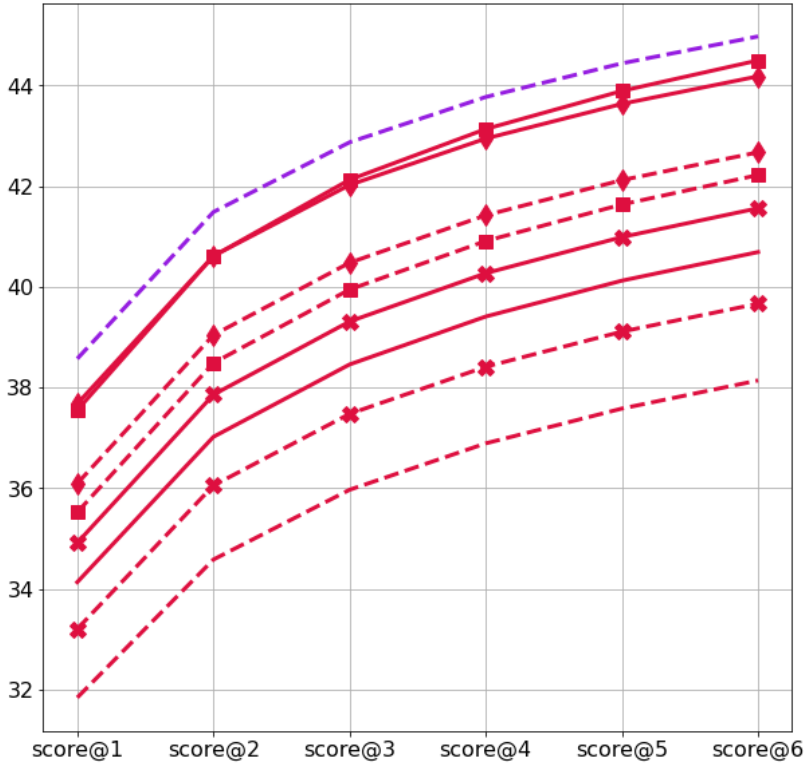
Improvements in Intent Satisfaction



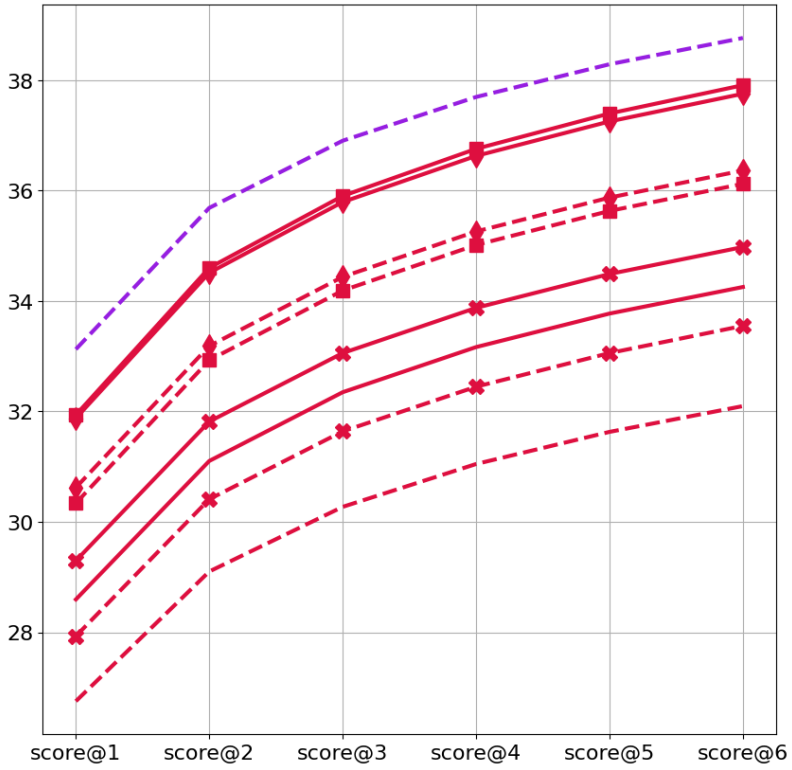
Next Identifier Match



Identifier Sequence Match

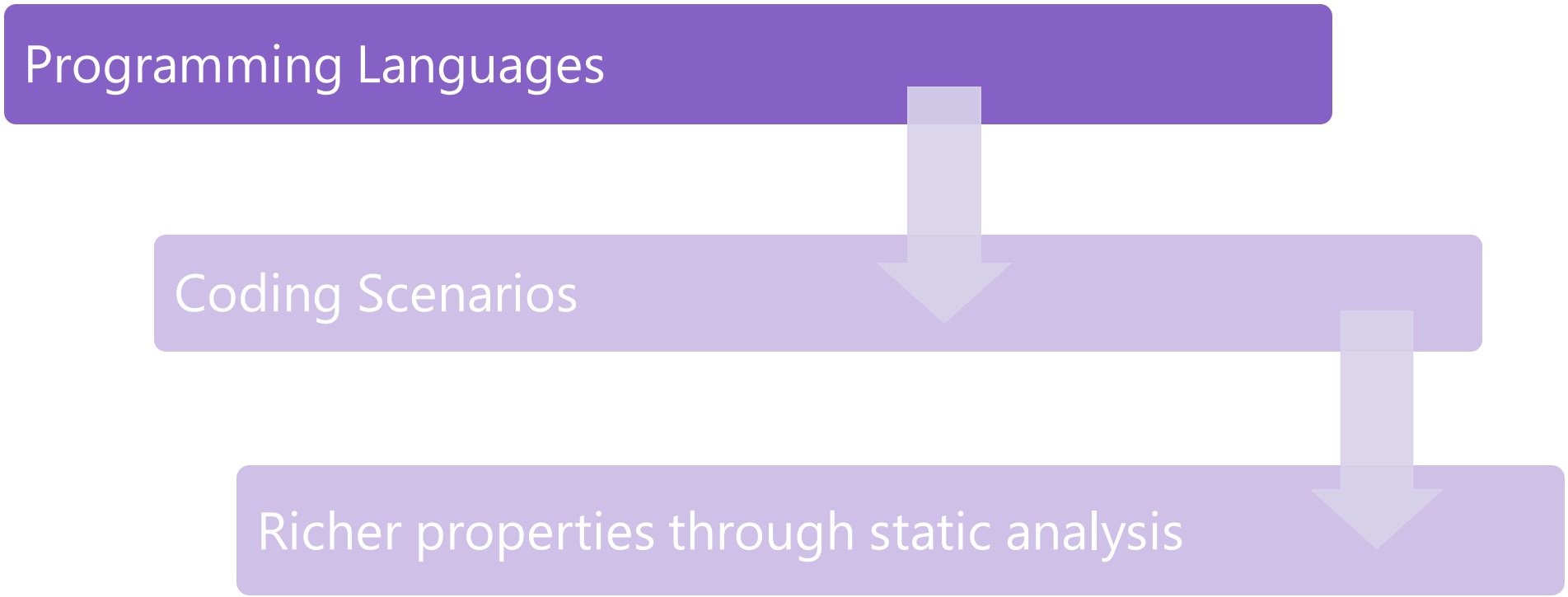


Prefix Match



SantaCoder-1.1B with MGD can generate type-correct non-local identifiers having better match with ground truth than text-davinci-003 (175B) across all tested numbers of trials

Generalizability study: MGDMicroBench



Generalizability study: Coding Scenarios

Correct number of arguments to methods (trigger on '<ident>(')

```
public ClickHouseDataSource withConnectionsCleaning(int rate, TimeUnit timeUnit) {
    this.driver.scheduleConnectionsCleaning( /* Monitor Triggers */
                void ru.yandex.clickhouse.ClickHouseDriver.scheduleConnectionsCleaning(int rate, TimeUnit timeUnit)
```

Valid class instantiation (trigger on 'new ')

```
private ClickHouseSinkBuffer(ClickHouseWriter writer) {
    this.writer = writer;
    this.localBuffer = Collections.synchronizedList(new  /* Monitor Triggers */
                                                    List                    java.util
                                                    ArrayList               java.util
                                                    COWArrayList            ch.qos.logback.core.util
```

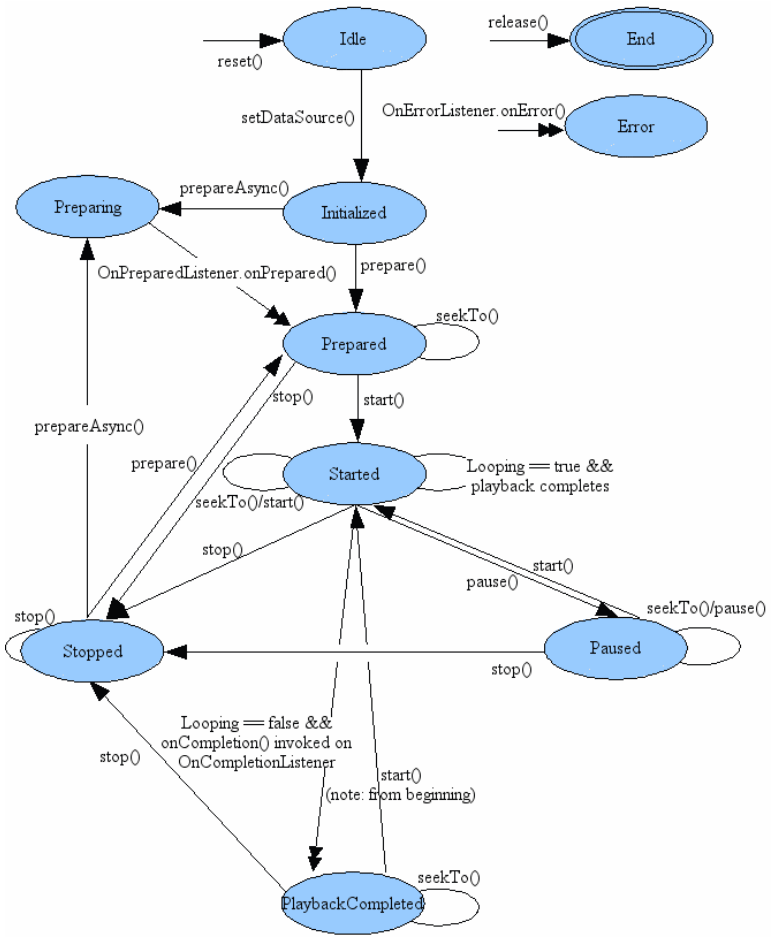
switch over enum (trigger on 'case ')

```
/* SpecialColors.java */
enum SpecialColors {
    SkyBlue, CrimsonRed, LeafyGreen
}
```

```
/* Main.java */
void printColorName(SpecialColors color) {
    switch(color) {
        case  /* Monitor triggers */
              CrimsonRed
              LeafyGreen
              SkyBlue
```

Generalizability study: Richer properties through static analysis

Typestate specifications, often expressed as finite state machines (FSMs), define valid sequences of operations that can be invoked on objects of a given type.




```
pub fn play_song(path: String) {
    let mut media_player = create_music_player();
    let media_player = media_player./* Monitor Triggers */
```

```
into() (as Into)
reset()
set_media_file_path(…)
```

```
pub fn play_song(path: String) {
    let mut media_player = create_music_player();
    let media_player = media_player.set_media_file_path(path);
    let media_player = media_player./* Monitor Triggers */
```

```
into() (as Into)
prepare()
reset()
```

Microsoft Microsoft Research

NEURAL INFORMATION PROCESSING SYSTEMS

Thank you!

Monitor-Guided Decoding of Code LMs with Static Analysis of Repository Context

Lakshya A Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K. Lahiri, Sriram Rajamani

Code and Data
aka.ms/monitors4codegen

Link to Paper
aka.ms/mgd_paper