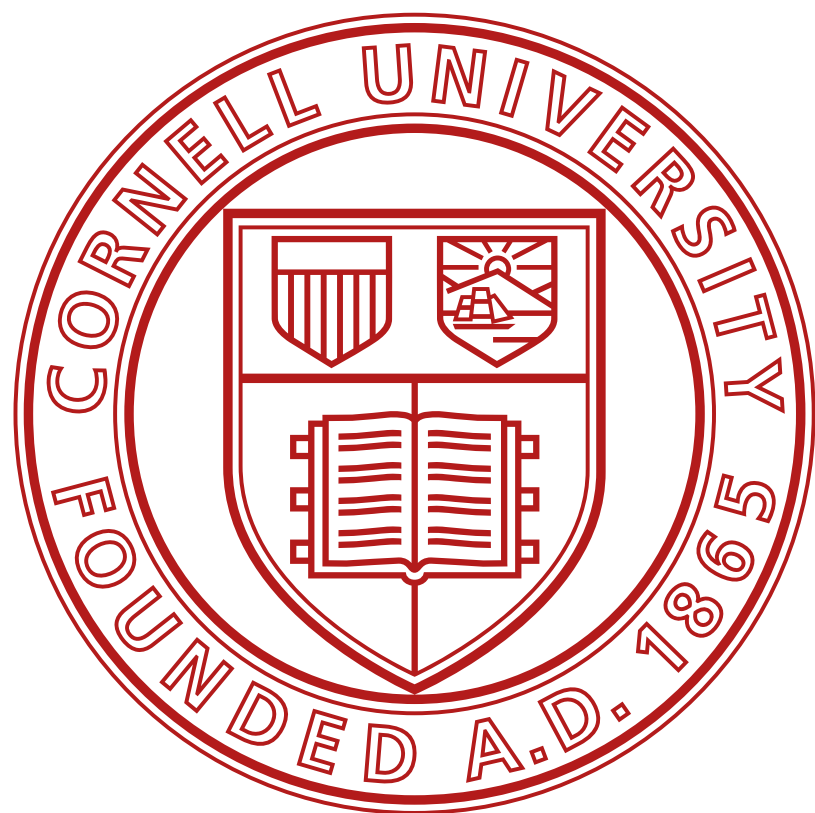


# Faster Runtime Verification during Testing via Feedback-Guided Selective Monitoring

**Shinhae Kim**, Saikat Dutta, and Owolabi Legunsen  
**Cornell University**



**Nov. 18th, 2025 @ ASE**



CCF-2045596, CCF-2319473, CCF-2403035, CCF-2525243

# What is Runtime Verification (RV)?

- Monitors program executions against formal specifications (specs)

# What is Runtime Verification (RV)?

- Monitors program executions against formal specifications (specs)

**Spec:** Appendable\_TheadSafe

“Two threads should not access the same Appendable object”

 “Internal values can be corrupted”

# What is Runtime Verification (RV)?

- Monitors program executions against formal specifications (specs)

**Spec:** Appendable\_TheadSafe

**Event:** safe\_append

- append(..) is called by the same thread T

**Event:** unsafe\_append

- append(..) is called by a thread T' s.t.,  $T' \neq T$

**unsafe\_append\* → violation** 

# What is Runtime Verification (RV)?

- Monitors program executions against formal specifications (specs)

**Spec:** Appendable\_TheadSafe

**Event:** safe\_append

- append(..) is called by the same thread T

**Event:** unsafe\_append

- append(..) is called by a thread T' s.t., T' != T

**unsafe\_append\* → violation** 

```
private double eval(final String f_x, final double xi) throws .. {  
    String number;  
    for (int i = 0; i < f_x.length(); i++) {  
        final char character = f_x.charAt(i);  
        if (character >= '0' && character <= '9') {  
            hasNumber = true;  
            number += character;  
            if (i == (f_x.length() - 1)) {  
                ..  
            }  
        }  
    }  
}
```

<https://github.com/sbesada/java.math.expression.parser>



# What is Runtime Verification (RV)?

- Monitors program executions against formal specifications (specs)

**Spec:** Appendable\_TheadSafe

**Event:** safe\_append

- append(..) is called by the same thread T

**Event:** unsafe\_append

- append(..) is called by a thread T' s.t., T' != T

**unsafe\_append\* → violation** ⚠

```
private double eval(final String f_x, final double xi) throws .. {  
    String number;  
    for (int i = 0; i < f_x.length(); i++) {  
        final char character = f_x.charAt(i);  
        if (character >= '0' && character <= '9') {  
            hasNumber = true;  
            number += character;  
            if (i == (f_x.length() - 1)) {  
                number = new StringBuilder( )  
                    .append(number).append(character).toString( );  
            }  
        }  
    }  
}
```

<https://github.com/sbesada/java.math.expression.parser>

# What is Runtime Verification (RV)?

- Monitors program executions against formal specifications (specs)

**Spec:** Appendable\_TheadSafe

**Event:** safe\_append

- append(..) is called by the same thread T

**Event:** unsafe\_append

- append(..) is called by a thread T' s.t., T' != T

**unsafe\_append\* → violation** ⚠

```
private double eval(final String f_x, final double xi) throws .. {  
    String number;  
    for (int i = 0; i < f_x.length(); i++) {  
        final char character = f_x.charAt(i);  
        if (character >= '0' && character <= '9') {  
            hasNumber = true;  
            number += character;  
            if (i == (f_x.length() - 1)) {  
                number = new StringBuilder( )  
                    .append(number).append(character).toString( );  
            }  
        }  
    }  
}
```

■ (monitor for the StringBuilder instance) ←

<https://github.com/sbesada/java.math.expression.parser>

# What is Runtime Verification (RV)?

- Monitors program executions against formal specifications (specs)

**Spec:** Appendable\_TheadSafe

**Event:** safe\_append

- append(..) is called by the same thread T

**Event:** unsafe\_append

- append(..) is called by a thread T' s.t., T' != T

**unsafe\_append\* → violation** ⚠

```
private double eval(final String f_x, final double xi) throws .. {  
    String number;  
    for (int i = 0; i < f_x.length(); i++) {  
        final char character = f_x.charAt(i);  
        if (character >= '0' && character <= '9') {  
            hasNumber = true;  
            number += character;  
            if (i == (f_x.length() - 1)) {  
                number = new StringBuilder( )  
                    .append(number).append(character).toString( );  
            }  
        }  
    }  
}
```

 **Trace:** [safe\_append, safe\_append]

<https://github.com/sbesada/java.math.expression.parser>



# RV Incurs High Overhead

- Has detected **hundreds of bugs** regarding correct JDK API usage<sup>[1-3]</sup>

[1] Legunsen et al., “How Good are the Specs? A Study of the Bug-Finding Effectiveness of Existing Java API Specifications,” ASE 2016

[2] Legunsen et al., “How Effective are Existing Java API Specifications for Finding Bugs during Runtime Verification?,” JASE 2019

[3] Miranda et al., “Prioritizing Runtime Verification Violations,” ICST 2020

# RV Incurs High Overhead

- Has detected **hundreds of bugs** regarding correct JDK API usage<sup>[1-3]</sup>

“Is the iterator’s `next()` called only after its `hasNext()` returns true?”

“Is a synchronized collection being accessed safely within synchronization?”

“Is the `OutputStream` flushed before calling `toByteArray()` on the underlying `ByteArrayOutputStream`?”

[1] Legunsen et al., “How Good are the Specs? A Study of the Bug-Finding Effectiveness of Existing Java API Specifications,” ASE 2016

[2] Legunsen et al., “How Effective are Existing Java API Specifications for Finding Bugs during Runtime Verification?,” JASE 2019

[3] Miranda et al., “Prioritizing Runtime Verification Violations,” ICST 2020

# RV Incurs High Overhead

- Has detected **hundreds of bugs** regarding correct JDK API usage [1-3]

“Is the iterator’s `next()` called only after its `hasNext()` returns true?”

“Is a synchronized collection being accessed safely within synchronization?”

“Is the `OutputStream` flushed before calling `toByteArray()` on the underlying `ByteArrayOutputStream`?”

**Great! However..**

A recent study shows that **runtime overhead** can be as high as 5,000x compared to unit testing only, or 27 hours [4]

[1] Legunsen et al., “How Good are the Specs? A Study of the Bug-Finding Effectiveness of Existing Java API Specifications,” ASE 2016

[2] Legunsen et al., “How Effective are Existing Java API Specifications for Finding Bugs during Runtime Verification?,” JASE 2019

[3] Miranda et al., “Prioritizing Runtime Verification Violations,” ICST 2020

[4] Guan and Legunsen, “An In-Depth Study of Runtime Verification Overheads During Software Testing,” ISSTA 2024

# One Reason for High Overhead

- Runtime Verification creates and uses **numerous monitors**
  - Loops, multiple calls to a method, ...

# One Reason for High Overhead

- Runtime Verification creates and uses **numerous monitors**
  - Loops, multiple calls to a method, ...

**Event:** safe\_append

- append(..) is called by a thread T

**Event:** unsafe\_append

- append(..) is called by a thread T' s.t., T' != T

**unsafe\_append\* → violation** 

 **Trace:** [safe\_append, safe\_append]

```
private double eval(final String f_x, final double xi) throws .. {  
    String number;  
    for (int i = 0; i < f_x.length(); i++) {  
        final char character = f_x.charAt(i);  
        if (character >= '0' && character <= '9') {  
            hasNumber = true;  
            number += character;  
            if (i == (f_x.length() - 1)) {  
                ..  
            }  
        }  
    }  
}
```



# One Reason for High Overhead

- Runtime Verification creates and uses **numerous monitors**
  - Loops, multiple calls to a method, ...




## Event: safe\_append

- append(..) is called by a thread T

## Event: unsafe\_append

- append(..) is called by a thread T' s.t., T' != T

**unsafe\_append\* → violation** 

-  **Trace:** [safe\_append, safe\_append] // iteration 1
-  **Trace:** [safe\_append, safe\_append] // iteration 2
-  **Trace:** [safe\_append, safe\_append] // iteration 3

...

```
private double eval(final String f_x, final double xi) throws .. {  
    String number;  
    for (int i = 0; i < f_x.length(); i++) {  
        final char character = f_x.charAt(i);  
        if (character >= '0' && character <= '9') {  
            hasNumber = true;  
            number += character;  
            if (i == (f_x.length() - 1)) {  
                ..  
            }  
        }  
    }  
}
```

# One Reason for High Overhead

- Runtime Verification creates and uses **numerous monitors**
  - Loops, multiple calls to a method, ...

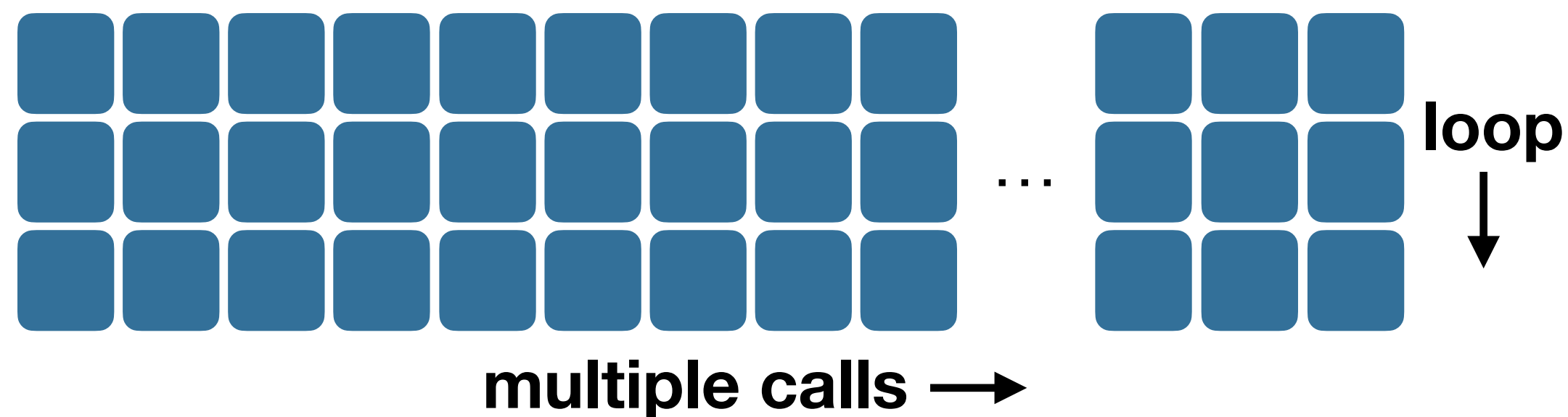
**Event:** safe\_append

- append(..) is called by a thread T

**Event:** unsafe\_append

- append(..) is called by a thread T' s.t., T' != T

**unsafe\_append\* → violation** ⚠



(other methods)  
`private double eval(final String f_x, final double xi) throws .. {  
 String number;`

`for (int i = 0; i < f_x.length(); i++) {`

`final char character = f_x.charAt(i);`

`if (character >= '0' && character <= '9') {`

`hasNumber = true;`

`number += character;`

`if (i == (f_x.length() - 1)) {`

`..`

**68,000,157 monitors** for the same trace

# One Reason for High Overhead

- Runtime Verification creates and uses **numerous monitors**
  - Loops, multiple calls to a method, ...

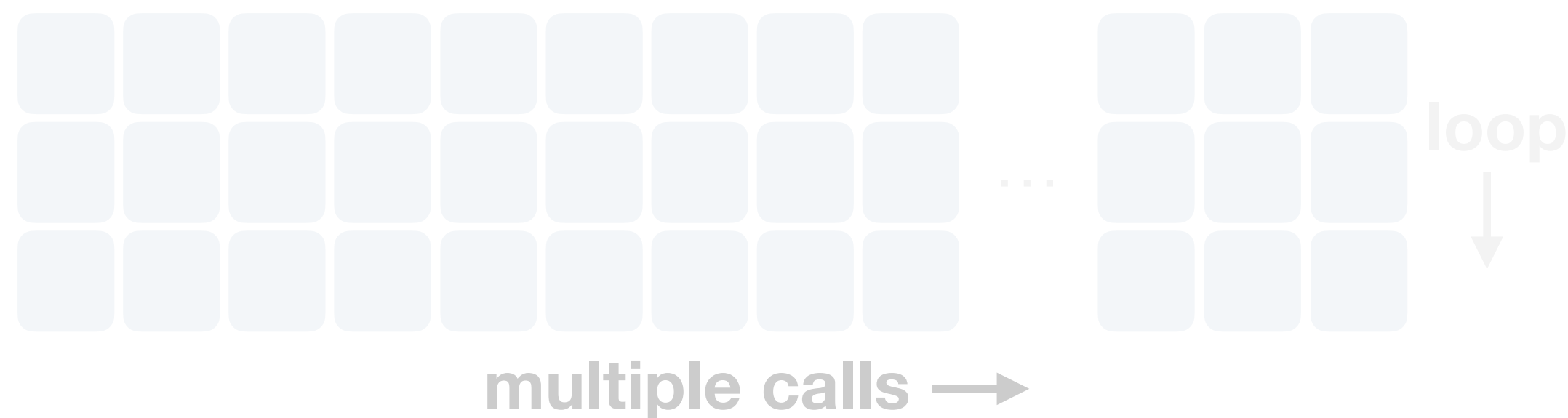
Event: safe\_append

- (+) RV monitors the executions of **multiple tests and loops** against **multiple specs**

Event: unsafe\_append

- append(..) is called by a thread T' s.t., T' != T

unsafe\_append → violation !



(other methods)  
private double eval(final String f\_x, final double xi) throws .. {

for (int i = 0; i < f\_x.length(); i++) {

final char character = f\_x.charAt(i);

if (character >= '0' && character <= '9') {

hasNumber = true;

number += character;

if (i == (f\_x.length() - 1)) {

..

**68,000,157 monitors** for the same trace

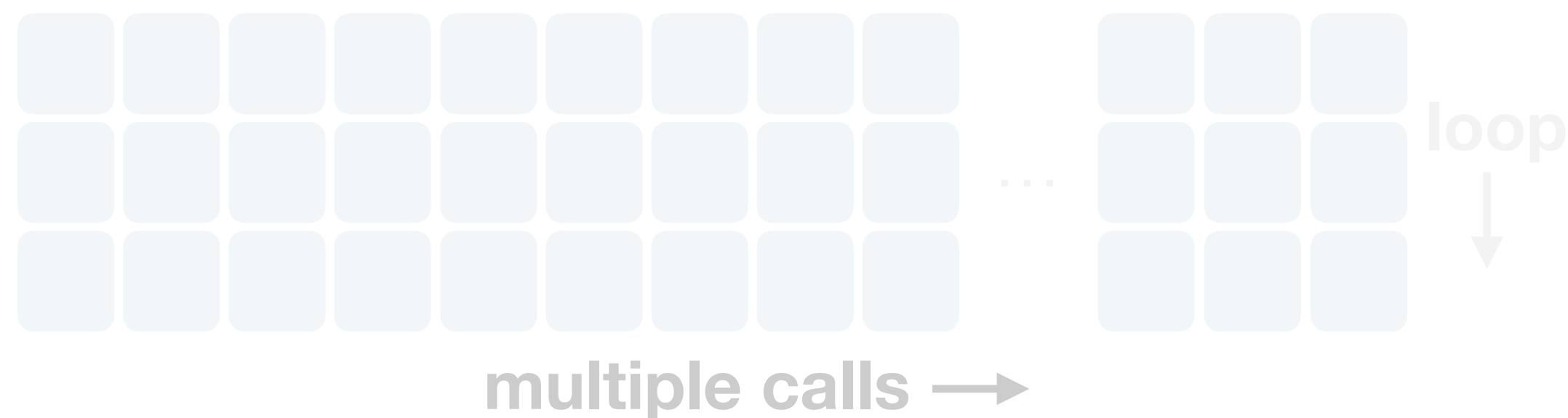
# One Reason for High Overhead

- Runtime Verification creates and uses **numerous monitors**
  - Loops, multiple calls to a method, ...

We present **Valg**, the first on-the-fly **selective monitoring** technique for RV and the first to use **reinforcement learning (RL)** to speed up RV

- `append(..)` is called by a thread `T'` s.t., `T' != T`

`unsafe_append` → violation ⚠



```
if (character >= '0' && character <= '9') {  
    hasNumber = true;  
    number += character;  
    if (i == (f_x.length() - 1)) {  
        ..  
    }  
}
```

**68,000,157 monitors** for the same trace

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations



# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

**Learning objective:** Reducing **redundant** monitors while preserving **unique** ones

**How?**

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

## 1) Formulate as two-armed bandit problem <sup>[5]</sup>

- **Actions:** {'create', 'ncreate'}

[5] Sutton and Barto, “Reinforcement Learning: An Introduction,” MIT Press, 2018

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

## 1) Formulate as two-armed bandit problem [5]

- **Actions:** {'create', 'ncreate'}
- **Binary reward** for 'create' and **continuous reward** for 'ncreate'

$$R_{\text{create},t} \doteq 0 \text{ if } (\text{trace}_t \text{ is redundant}) \text{ else } 1 \quad R_{\text{ncreate},t} \doteq \frac{\sum_{k=0}^{t-1} 1(\text{trace}_k \text{ is redundant})}{\sum_{k=0}^{t-1} 1(\text{trace}_k \text{ is observed})}$$

[5] Sutton and Barto, “Reinforcement Learning: An Introduction,” MIT Press, 2018

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

## 1) Formulate as two-armed bandit problem [5]

- **Actions:** {'create', 'ncreate'}
- **Binary reward** for 'create' and **continuous reward** for 'ncreate'

$$R_{\text{create},t} \doteq 0 \text{ if } (\text{trace}_t \text{ is redundant}) \text{ else } 1 \quad R_{\text{ncreate},t} \doteq \frac{\sum_{k=0}^{t-1} 1(\text{trace}_k \text{ is redundant})}{\sum_{k=0}^{t-1} 1(\text{trace}_k \text{ is observed})}$$

## 2) Select based on action-value method

- **Exponential-recency weighted average** to estimate rewards for each action [5]
- **Epsilon-greedy** to enable stochastic exploration (vs. exploitation) [5]

[5] Sutton and Barto, "Reinforcement Learning: An Introduction," MIT Press, 2018

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

## 1) Formulate as two-armed bandit problem [5]

- **Actions:** {'create', 'ncreate'}
- **Binary reward** for 'create' and **continuous reward** for 'ncreate'

$$R_{\text{create},t} \doteq 0 \text{ if } (\text{trace}_t \text{ is redundant}) \text{ else } 1 \quad R_{\text{ncreate},t} \doteq \frac{\sum_{k=0}^{t-1} 1(\text{trace}_k \text{ is redundant})}{\sum_{k=0}^{t-1} 1(\text{trace}_k \text{ is observed})}$$

**“Hyperparameters”**

## 2) Select based on action-value method

- **Exponential-recency weighted average** to estimate rewards for each action
- **Epsilon-greedy** to enable stochastic exploration (vs. exploitation)

Learning rate

Exploration prob.

[5] Sutton and Barto, “Reinforcement Learning: An Introduction,” MIT Press, 2018



# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

## 1) Formulate as two-armed bandit problem [5]

- **Actions:** {‘create’, ‘ncreate’}
- **Binary reward** for ‘create’ and **continuous reward** for ‘ncreate’

$$R_{\text{create},t} \doteq 0 \text{ if } (\text{trace}_t \text{ is redundant}) \text{ else } 1 \quad R_{\text{ncreate},t} \doteq \frac{\sum_{k=0}^{t-1} 1(\text{trace}_k \text{ is redundant})}{\sum_{k=0}^{t-1} 1(\text{trace}_k \text{ is observed})}$$

**“Hyperparameters”**

## 2) Select based on action-value method

Convergence threshold

Initial values

- **Exponential-recency weighted average** to estimate rewards for each action
- **Epsilon-greedy** to enable stochastic exploration (vs. exploitation)

Learning rate

Exploration prob.

[5] Sutton and Barto, “Reinforcement Learning: An Introduction,” MIT Press, 2018

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

■ @ iteration 1  
**Trace:** [safe\_append, safe\_append]

■ @ iteration 2  
**Trace:** [safe\_append, safe\_append]

■ @ iteration 3  
**Trace:** [safe\_append, safe\_append]

...

■ @ iteration 68,000,157  
**Trace:** [safe\_append, safe\_append]

[create] ■ @ iteration 1  
**Trace:** [safe\_append, safe\_append] 

**State-of-the-Art (SoTA) RV**

**Valg**

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

 @ iteration 1  
**Trace:** [safe\_append, safe\_append]

 @ iteration 2  
**Trace:** [safe\_append, safe\_append]



 @ iteration 3  
**Trace:** [safe\_append, safe\_append]

...

 @ iteration 68,000,157  
**Trace:** [safe\_append, safe\_append]

**State-of-the-Art (SoTA) RV**

[create]  @ iteration 1  
**Trace:** [safe\_append, safe\_append]

[create]  @ iteration 2  
**Trace:** [safe\_append, safe\_append] 

**Valg**

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

 @ iteration 1  
**Trace:** [safe\_append, safe\_append]

 @ iteration 2  
**Trace:** [safe\_append, safe\_append]

 @ iteration 3  
**Trace:** [safe\_append, safe\_append]

...

 @ iteration 68,000,157  
**Trace:** [safe\_append, safe\_append]

**State-of-the-Art (SoTA) RV**

[create]  @ iteration 1  
**Trace:** [safe\_append, safe\_append]

[create]  @ iteration 2  
**Trace:** [safe\_append, safe\_append]

[ncreate]  @ iteration 3  
**Trace:** [safe\_append, safe\_append]

**Valg**

# High-Level Overview of Valg

- Attaches a **reinforcement learning agent** to monitor creation location
- **Selectively** creates monitors based on past trace observations

 @ iteration 1  
**Trace:** [safe\_append, safe\_append]

 @ iteration 2  
**Trace:** [safe\_append, safe\_append]

 @ iteration 3  
**Trace:** [safe\_append, safe\_append]

...

 @ iteration 68,000,157  
**Trace:** [safe\_append, safe\_append]

**68,000,157 monitors**

[create]  @ iteration 1  
**Trace:** [safe\_append, safe\_append]

[create]  @ iteration 2  
**Trace:** [safe\_append, safe\_append]

[ncreate]  @ iteration 3  
**Trace:** [safe\_append, safe\_append]

...

[ncreate]  @ iteration 68,000,157  
**Trace:** [safe\_append, safe\_append]

**2 monitors**



# More Technical Details in Paper

- Rationale for the algorithm selection
- Convergence logic
- Initial value selection
- **Selective monitoring for another class of specs**
- Handling of ‘partially-bound’ monitors
- Valg-related optimizations
- Many interesting discussions

# Evaluation (Highlights)

**RQ1:** How does Valg compare with SoTA techniques?

## Experiment Setup

64 Java open-source projects with 160 JDK API specs

**Baselines:** Two SoTA techniques (JavaMOP<sup>[6]</sup> and TraceMOP<sup>[6]</sup>)

**RQ2:** How does Valg compare with random sampling approaches?

[6] “TraceMOP: A Trace-Aware Runtime Verification Tool for Java,” 2024, <https://github.com/SoftEngResearch/tracemop>

# Evaluation (Highlights)

**RQ1:** How does Valg compare with SoTA techniques?

## Experiment Setup

64 Java open-source projects with 160 JDK API specs

**Baselines:** Two SoTA techniques (JavaMOP<sup>[6]</sup> and TraceMOP<sup>[6]</sup>)

- Valg is up to **20.2x (4.3 hrs)** and **551.5x (24.3 hrs)** faster
- Valg reduces **3.02 days** down to **11.6 minutes** for three projects in total
- Valg preserves **99.6%** of unique violations detected by the baseline

[6] “TraceMOP: A Trace-Aware Runtime Verification Tool for Java,” 2024, <https://github.com/SoftEngResearch/tracemop>

# Evaluation (Highlights)

**RQ3:** What is the impact of hyperparameter tuning?

**RQ4:** How effective and efficient is Valg as code evolves?

## Experiment Setup

Valg with **default** hyperparameters vs. **tuned** hyperparameters  
1,472 versions of 46 projects for evolution experiments

# Evaluation (Highlights)

**RQ3:** What is the impact of hyperparameter tuning?

**RQ4:** How effective and efficient is Valg as code evolves?

## Experiment Setup

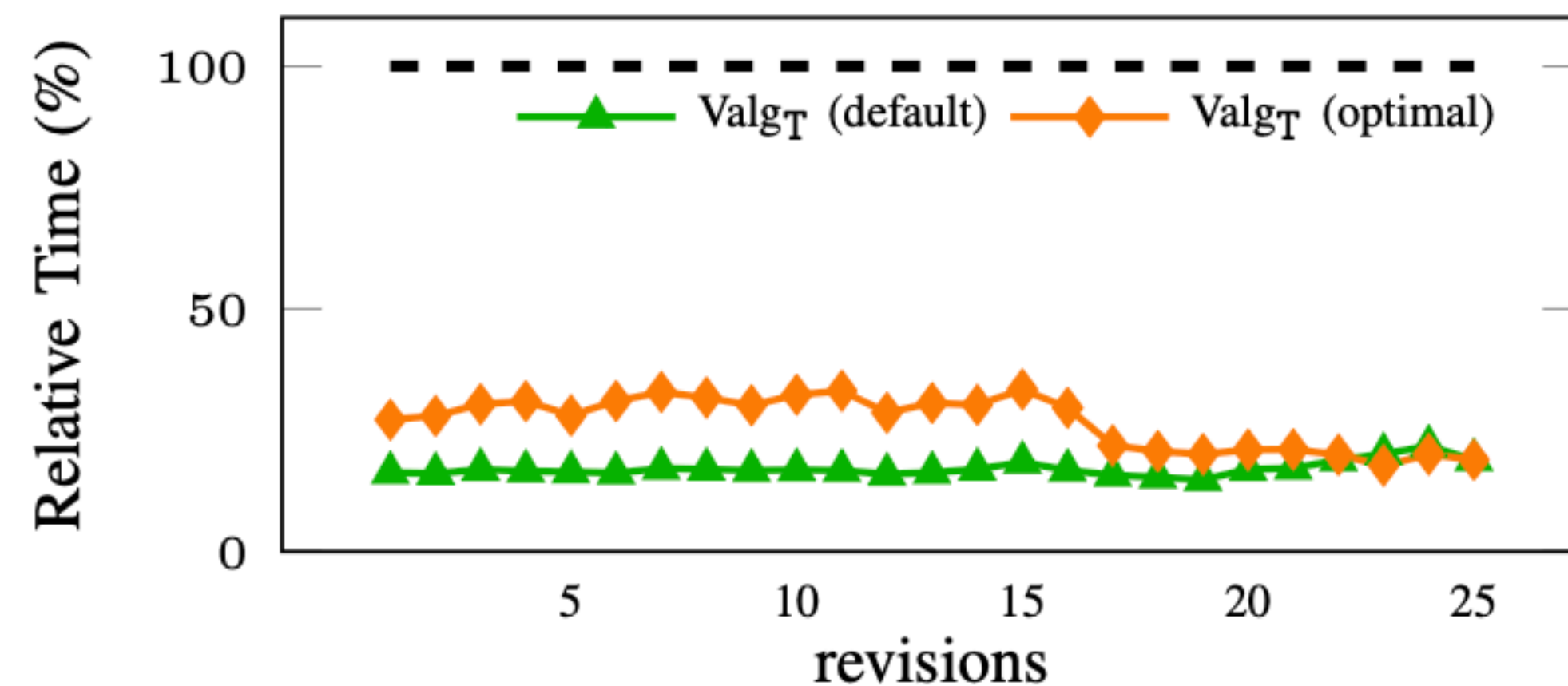
Valg with **default** hyperparameters vs. **tuned** hyperparameters  
1,472 versions of 46 projects for evolution experiments

- Valg's unique trace preservation ratio improves from **76.7%** to **95.1%**
- Tuned hyperparameters preserve Valg's effectiveness as programs evolve

# Evaluation (Highlights)

**RQ3:** What is the impact of hyperparameter tuning?

**RQ4:** How effective and efficient is Valg as code evolves?

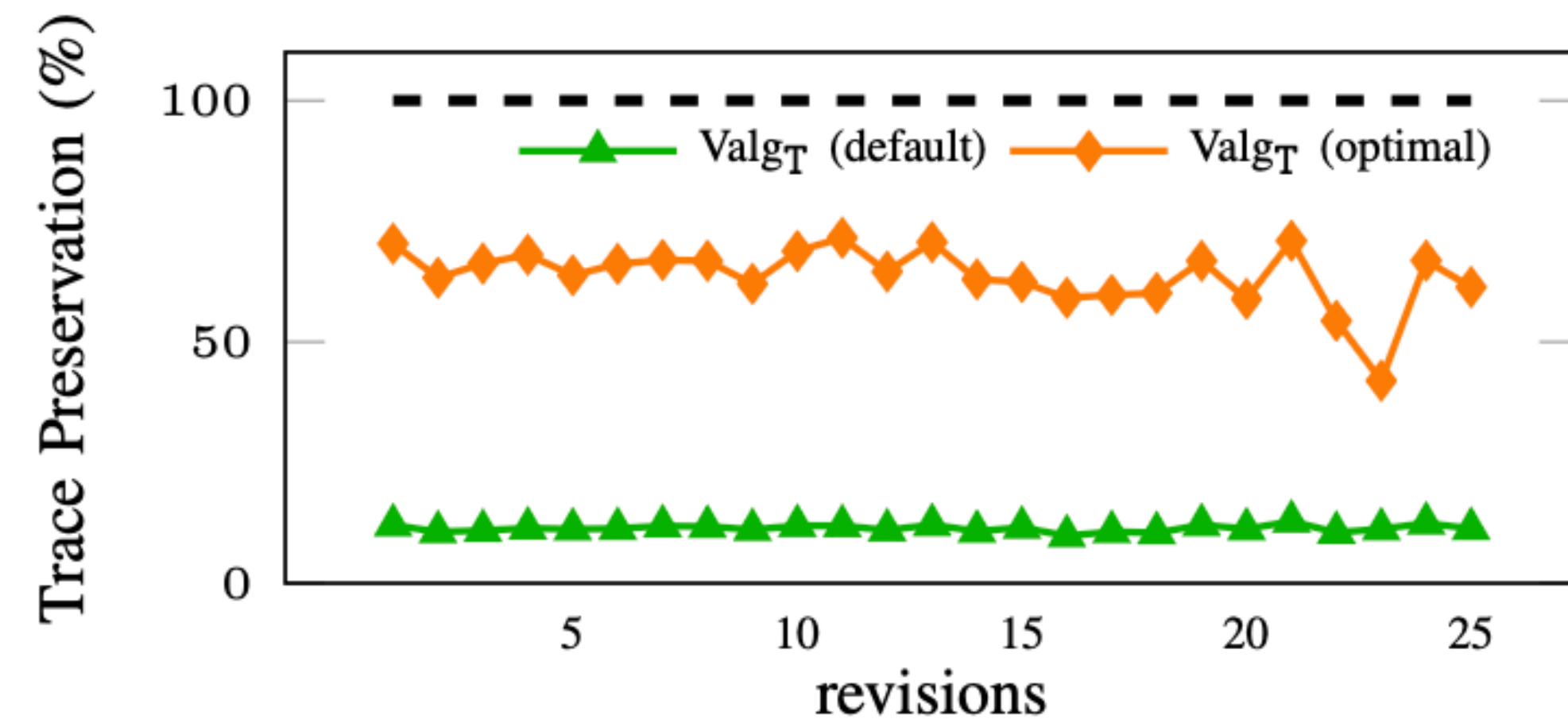
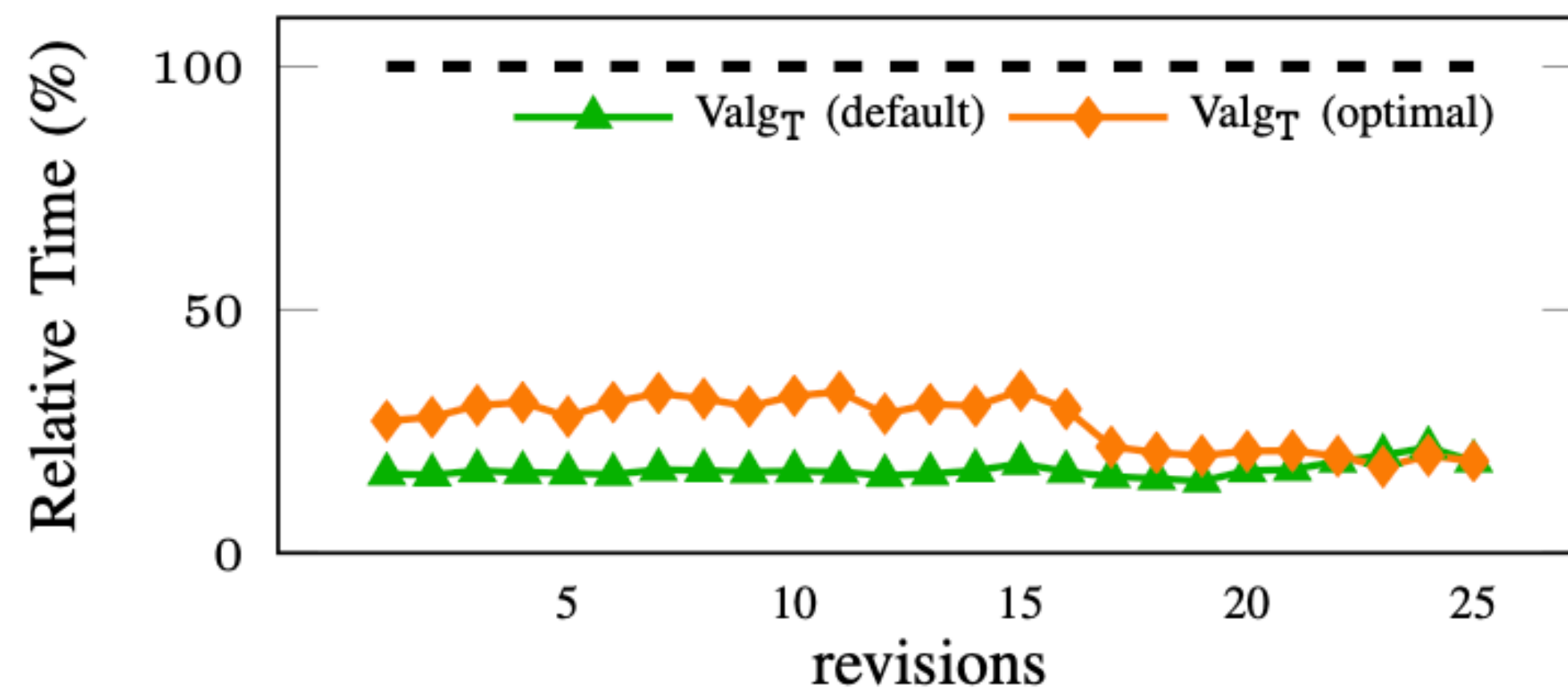


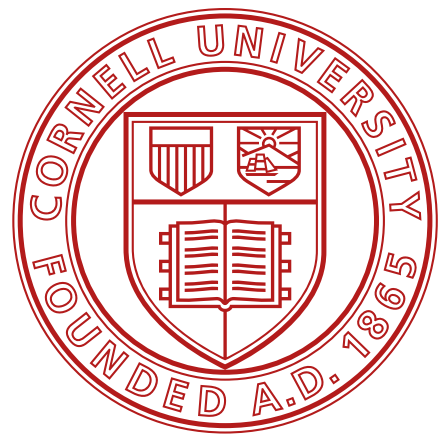


# Evaluation (Highlights)

**RQ3:** What is the impact of hyperparameter tuning?

**RQ4:** How effective and efficient is Valg as code evolves?





# Conclusion

We present **Valg**, the first on-the-fly **selective monitoring** technique for RV and the first to use **reinforcement learning** to speed up RV

- Valg is significantly **faster** than all baselines while preserving **violations**
- Hyperparameter tuning improves **unique trace preservation** with little cost
- The tuning's effectiveness is preserved across future versions

Artifact

<https://github.com/SoftEngResearch/Valg>

Contact

sk3364@cornell.edu

