# Wendigo

## Deep Reinforcement Learning for Denial-of-Service Query Discovery in GraphQL

**Shae McFadden**[1,2], Marcello Maugeri[3,1], Chris Hicks[2], Vasilios Mavroudis[2], Fabio Pierazzi[1]
[1]King's College London, [2]The Alan Turing Institute, [3]University of Catania

Supported by

# Example Scenario

## REST vs GraphQL

Imagine…

A third party application using an API to access information from a Web service.

And…

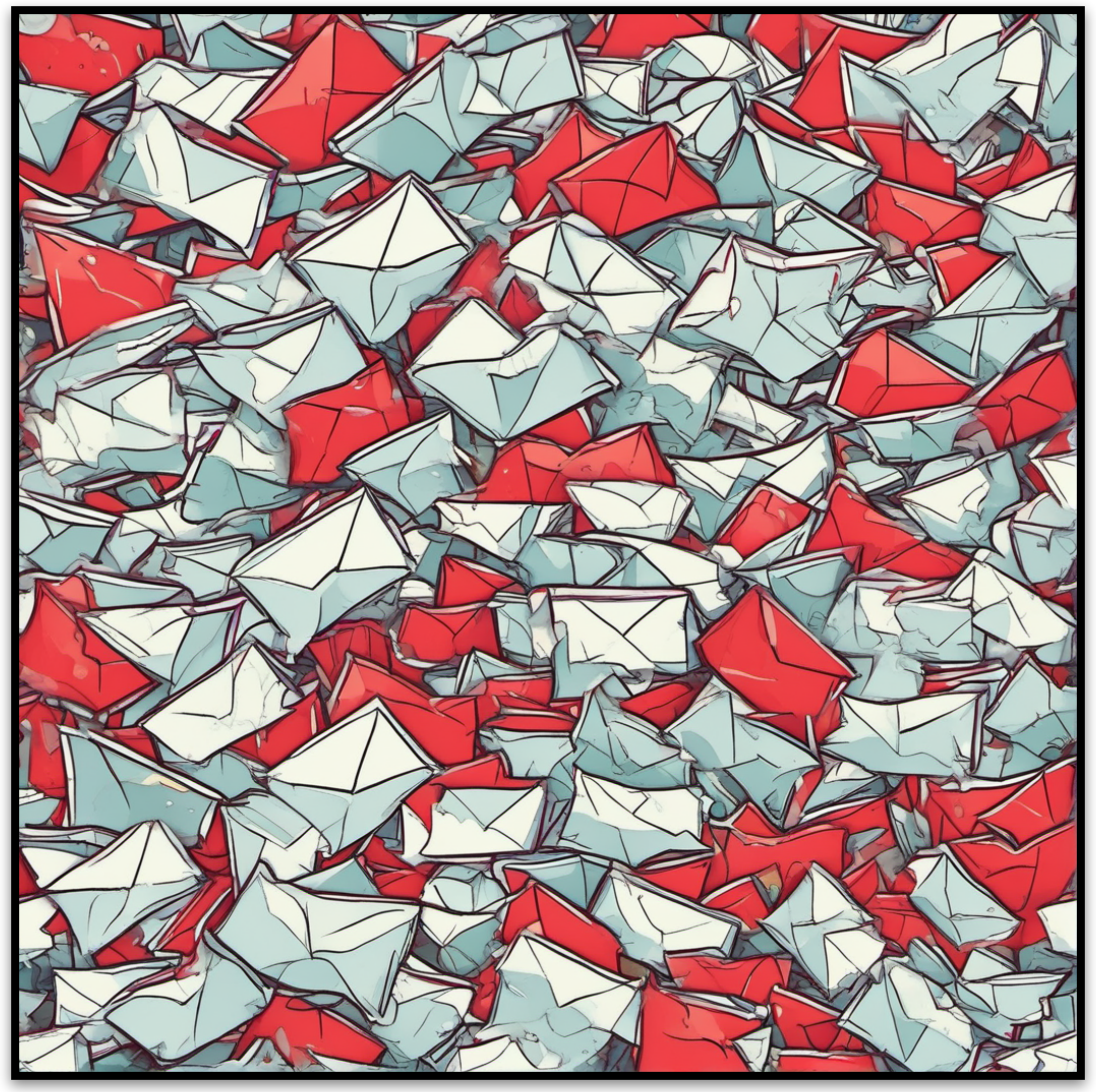
You want to retrieve the email and content of posts from a specific user.

# REST APIs

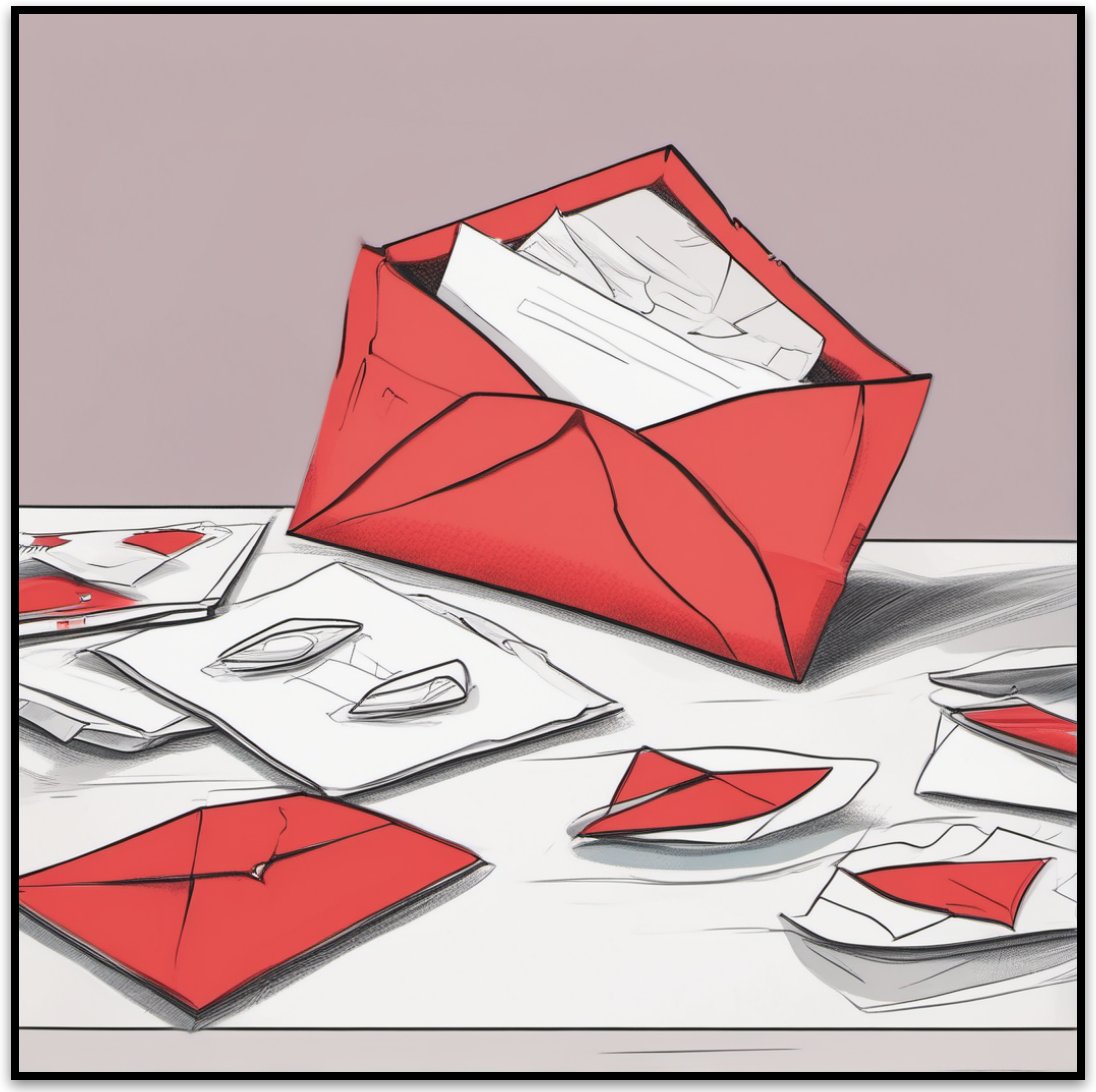
## Background

**REST**

1. *users/{id}*
2. *users/{id}/content*



**Over-Fetching**

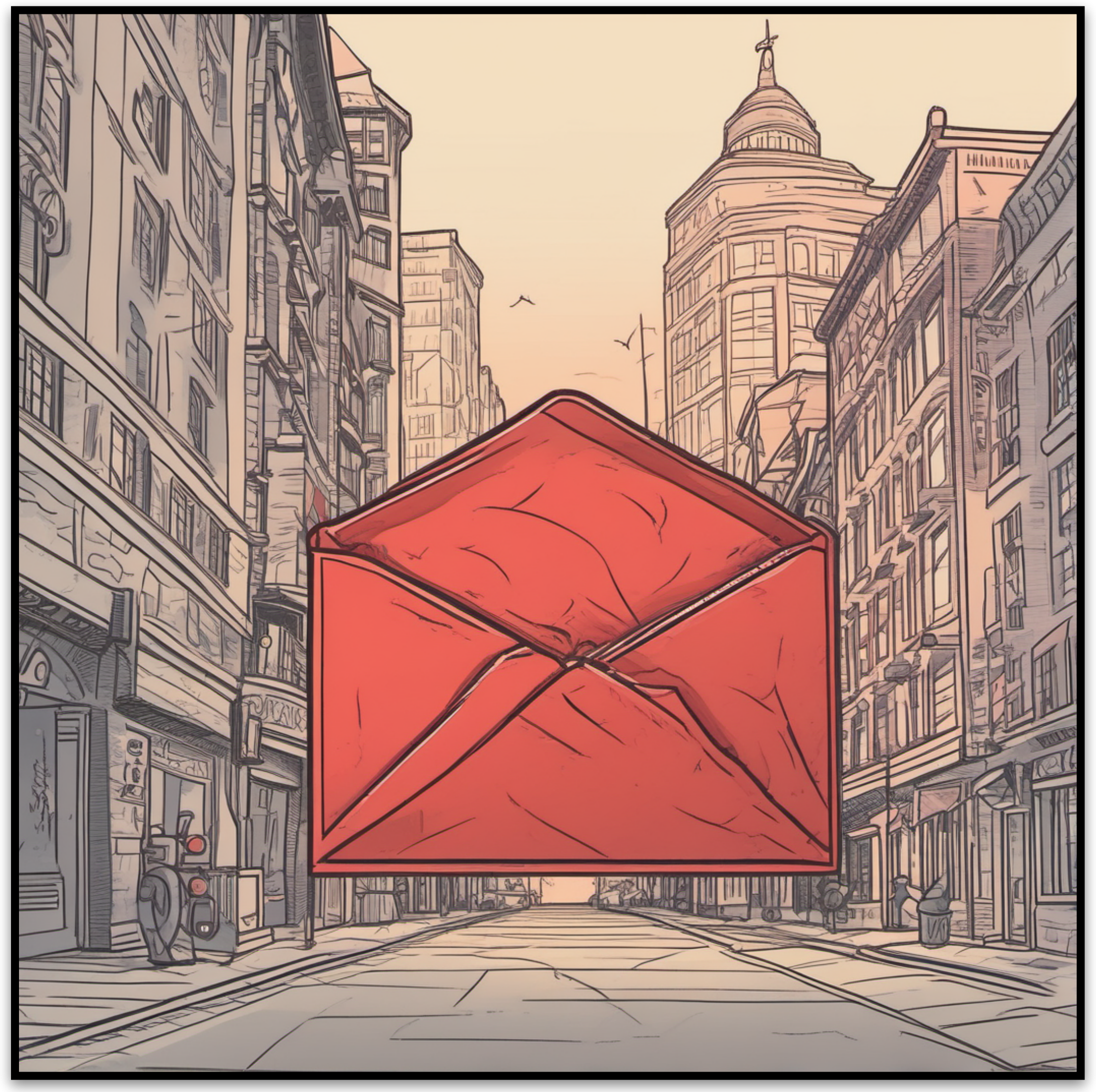


**Under-Fetching**

Image Credit: SDXL-Lightning

# GraphQL APIs

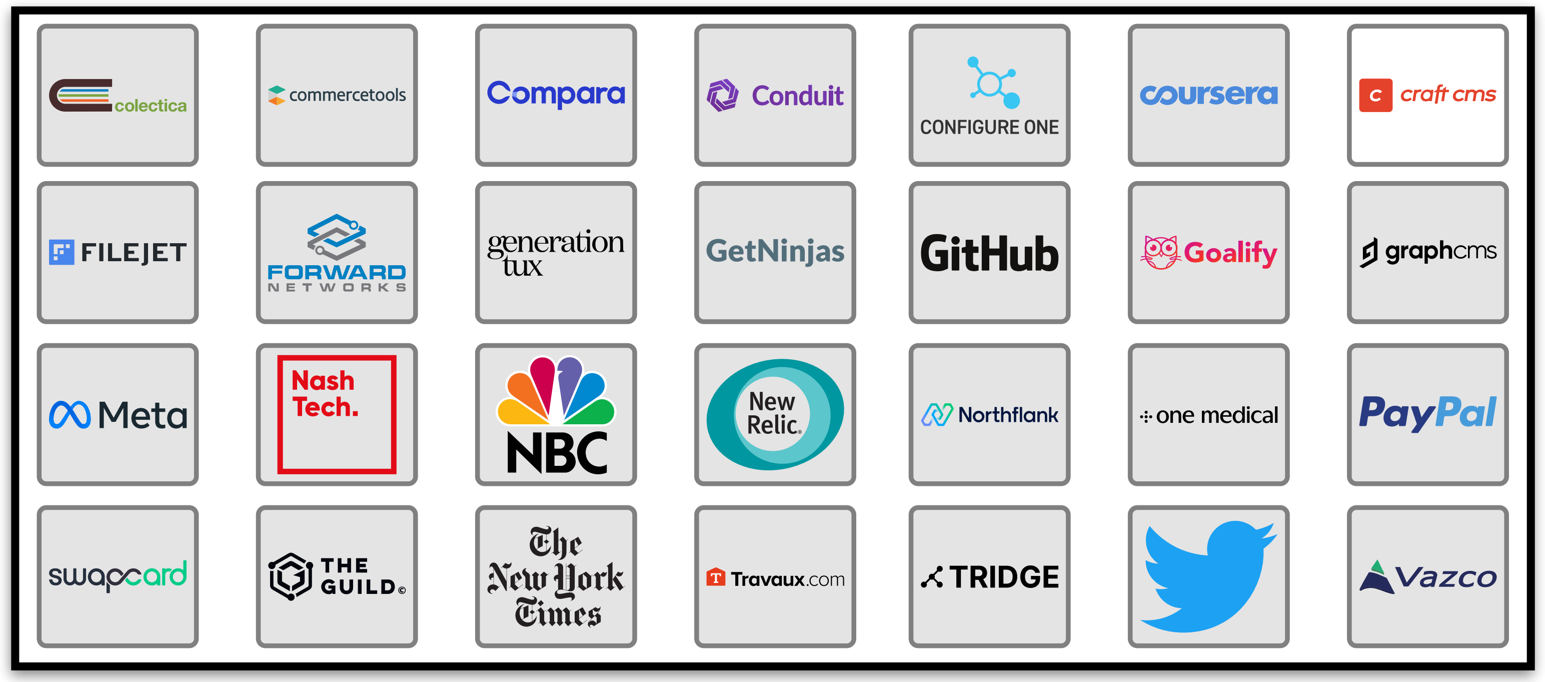
## Background



“Just What You Need”
-Fetching

**GraphQL**

```
query {
  users (id: {id}) {
    email
    pastes{
      content
    } …
```



**GraphQL Adopters**

Adopters Source: https://landscape.graphql.org/

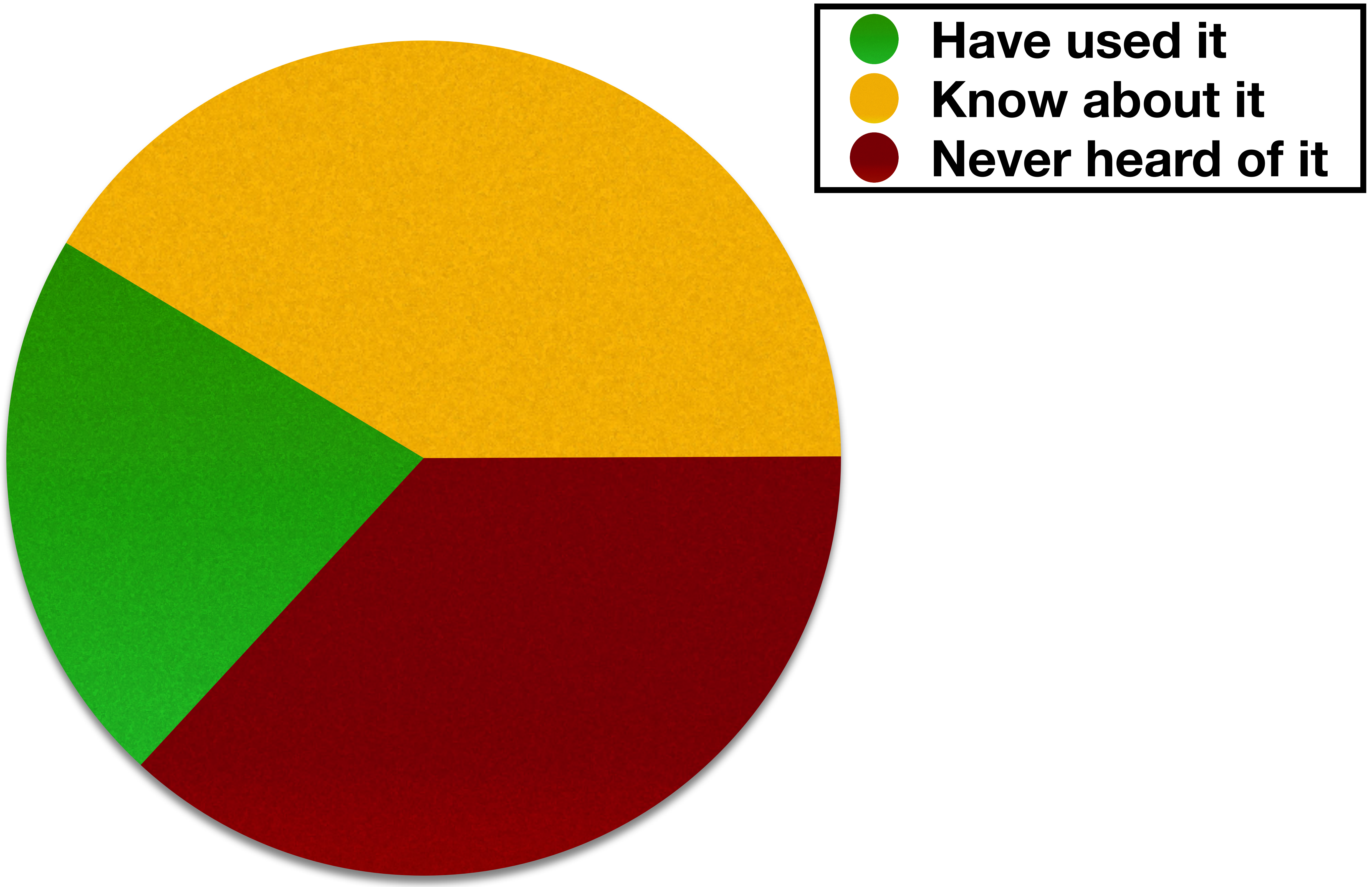
Image Credit: SDXL-Lightning

“Most commercial and large open-source GraphQL APIs may be susceptible to [DoS] queries”[1]

[1] Wittern, Erik, et al. "An empirical study of GraphQL schemas”. ICSOC 2019.

# Usage of DoS Defences

## GraphQL



Averaged results from a 2022 survey
of over two-thousand developers.

Data Source: https://2022.stateofgraphql.com/

# Denial-of-Service

## An Availability Attack

### Denial-of-Service (DoS)

*Utilizes a high volume of traffic to overwhelm the target.*

### <u>Low Rate DoS (LDoS)</u>

*Utilizes pulses of traffic to cause bottlenecks.*[2]

[2] W. Zhijun, et al. “Low-rate dos attacks, detection, defense, and challenges: A survey”

Image Credit: SDXL-Lightning

# Many past studies discuss the risk of DoS in GraphQL[1,3]… mentioning handcrafted queries.

[1] Wittern, Erik, et al. "An empirical study of GraphQL schemas". ICSOC 2019.

[3] Brito, Gleison, et al. "Migrating to GraphQL: A practical assessment". SANER 2019.

# Our Approach

**Leverage**

The capabilities of deep reinforcement learning

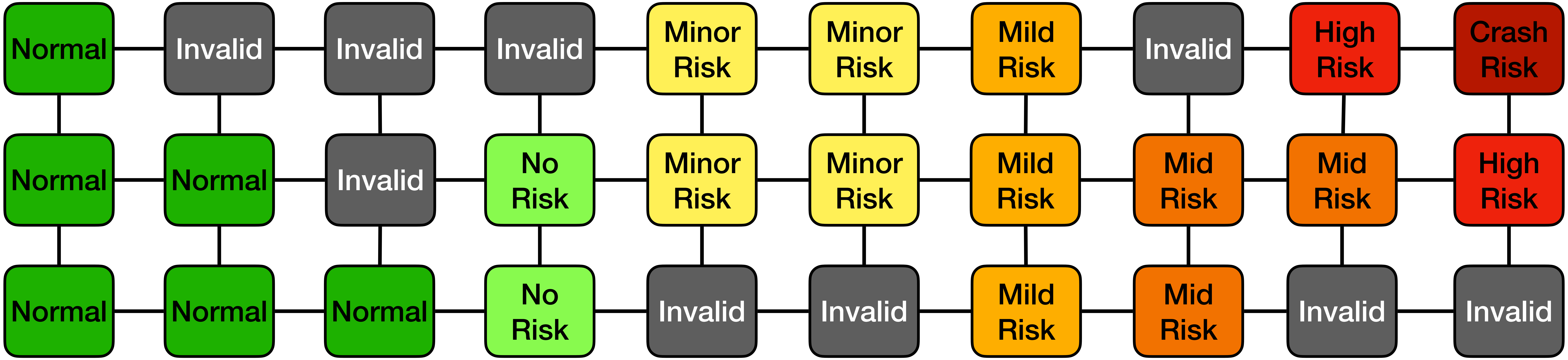
**Automate**

The creation and search of GraphQL queries

**Discover**

Queries that pose a LDoS risk

## The Potential Query Space



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Normal | Invalid | Invalid | Invalid | Minor Risk | Minor Risk | Mild Risk | Invalid | High Risk | Crash Risk |
| Normal | Normal | Invalid | No Risk | Minor Risk | Minor Risk | Mild Risk | Mid Risk | Mid Risk | High Risk |
| Normal | Normal | Normal | No Risk | Invalid | Invalid | Mild Risk | Mid Risk | Invalid | Invalid |

# Wendigo

## Our Approach

**Wendigo**

A DRL-based black-box approach for DoS query discovery.

**Duplication Capabilities**

Field Duplication

Alias Overloading

Array-Based Query Batching

**Retrieval Capabilities**

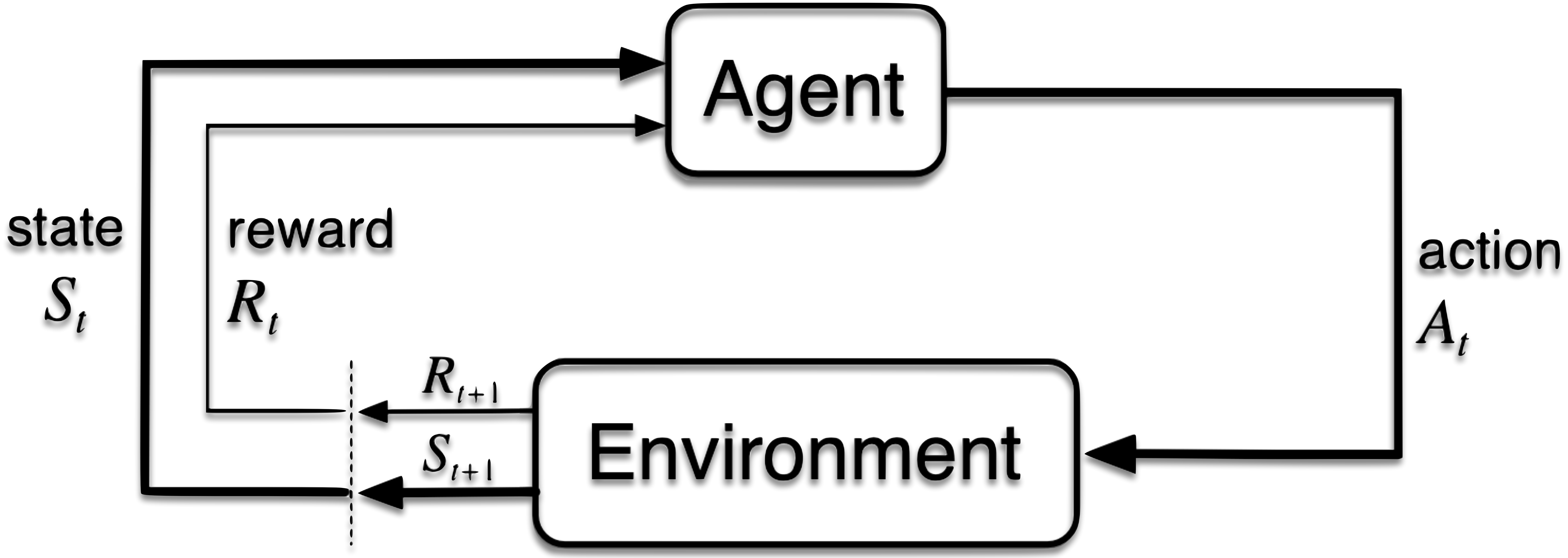
Object Limit Overriding

Circular Objects

Attack Vector Source: N. Aleks, et al. Black Hat GraphQL: Attacking Next Generation APIs.

# Deep Reinforcement Learning

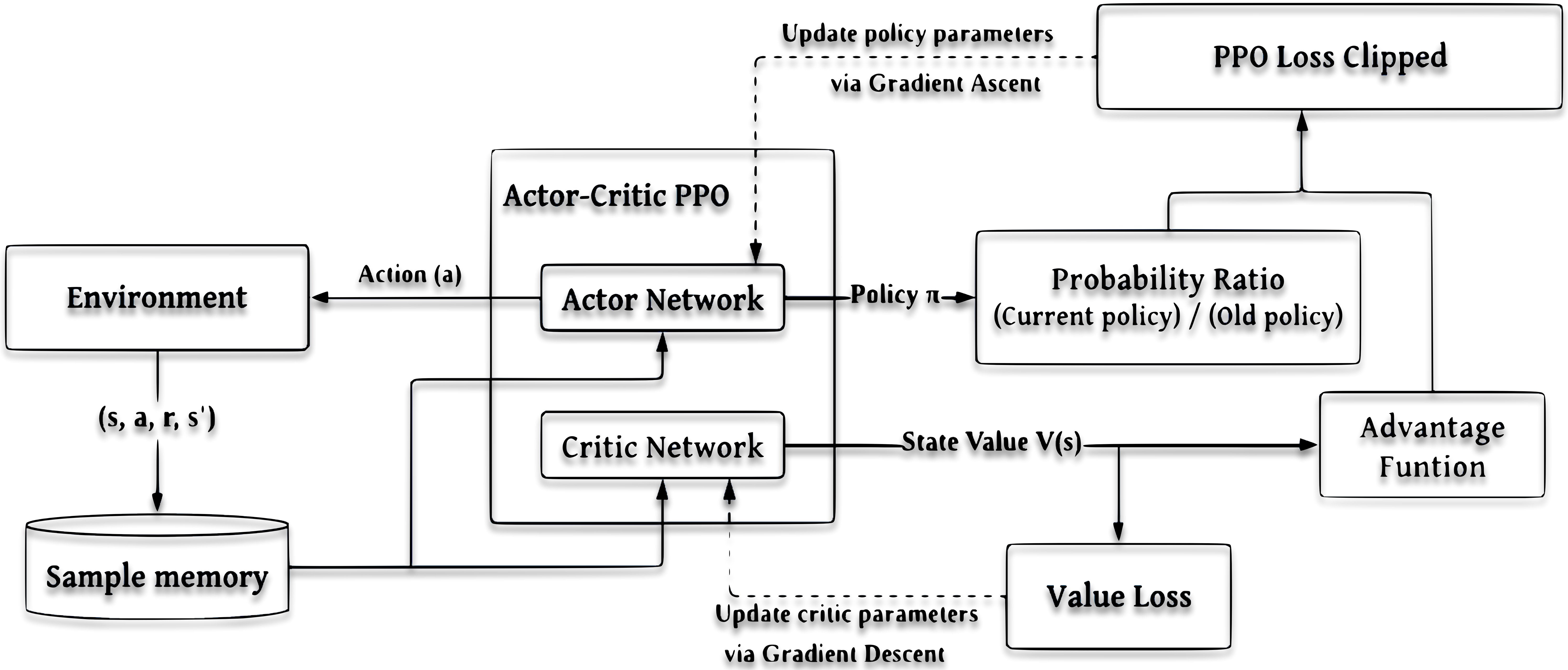
## Preliminary



Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

# Proximal Policy Optimization

## Agent



Kalidas, A. P., et al. "Deep reinforcement learning for vision-based navigation of UAVs in avoiding stationary and mobile obstacles." Drones (2023)

# Black Box

## Threat Model

**Constraints**

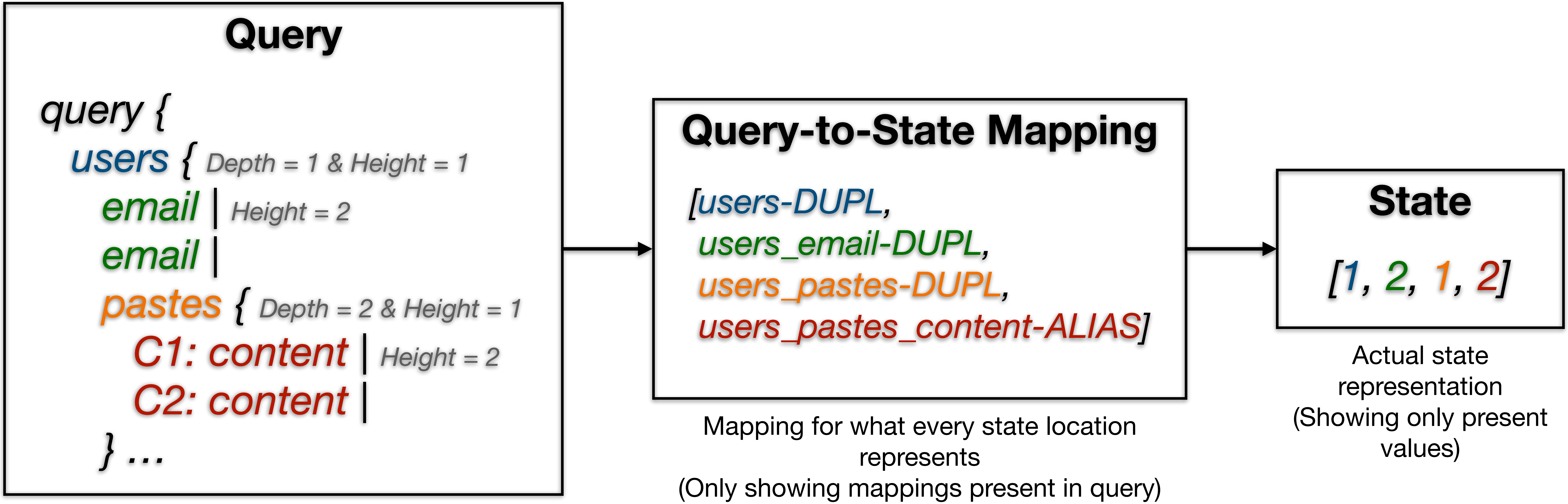
Only requires the schema & connection for an application.

**Motivation**

Enables plug-and-play security testing.

# States

## Environment



**Query**

```
query {
  users { Depth = 1 & Height = 1
    email | Height = 2
    email |
    pastes { Depth = 2 & Height = 1
      C1: content | Height = 2
      C2: content |
  } …
```

Example query (Displaying the height and depth of each location)

**Query-to-State Mapping**

[users-DUPL,
users_email-DUPL,
users_pastes-DUPL,
users_pastes_content-ALIAS]

Mapping for what every state location represents
(Only showing mappings present in query)

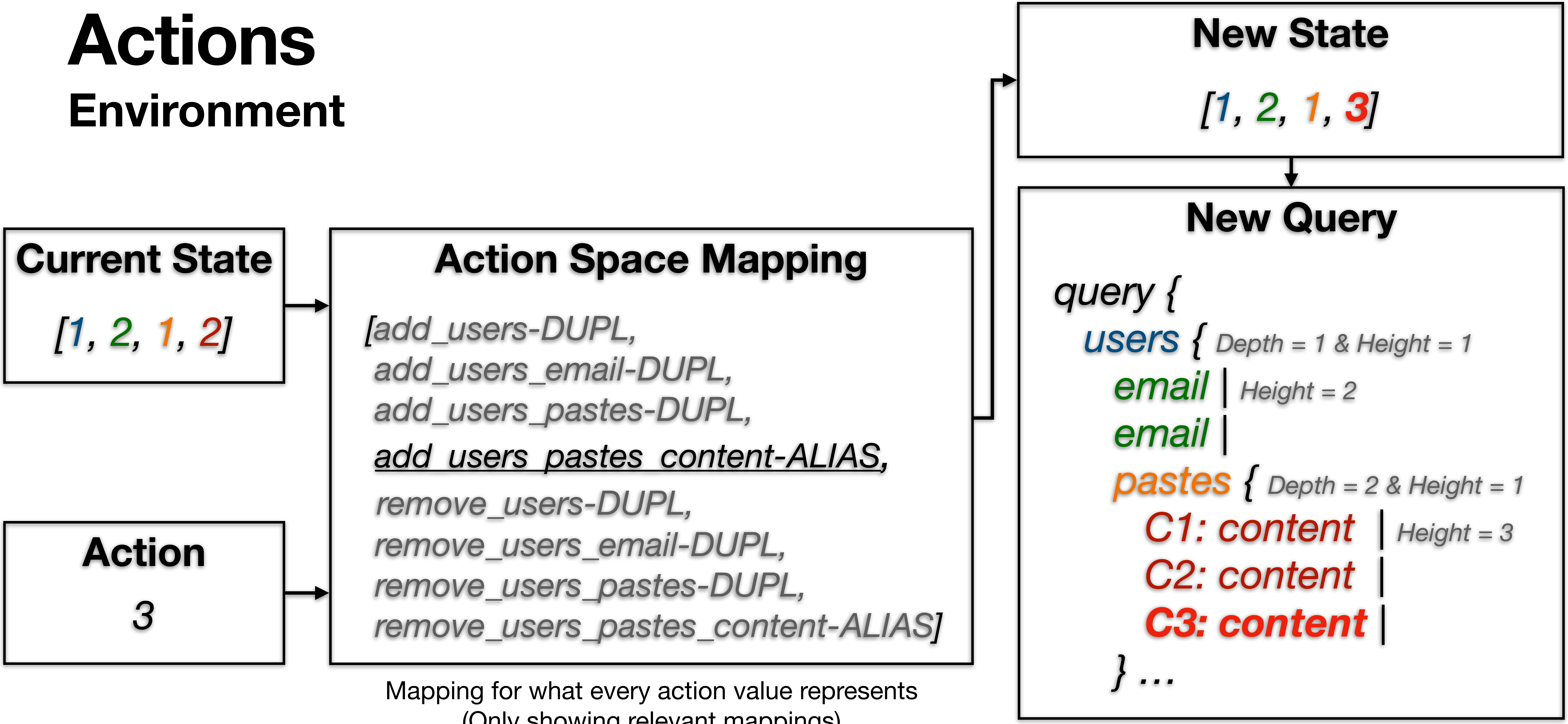
**State**

[1, 2, 1, 2]

Actual state representation
(Showing only present values)

# Actions

## Environment



**Current State**

[1, 2, 1, 2]

**Action**

3

**Action Space Mapping**

[add_users-DUPL,
add_users_email-DUPL,
add_users_pastes-DUPL,
add_users_pastes_content-ALIAS,
remove_users-DUPL,
remove_users_email-DUPL,
remove_users_pastes-DUPL,
remove_users_pastes_content-ALIAS]

Mapping for what every action value represents
(Only showing relevant mappings)

**New State**

[1, 2, 1, **3**]

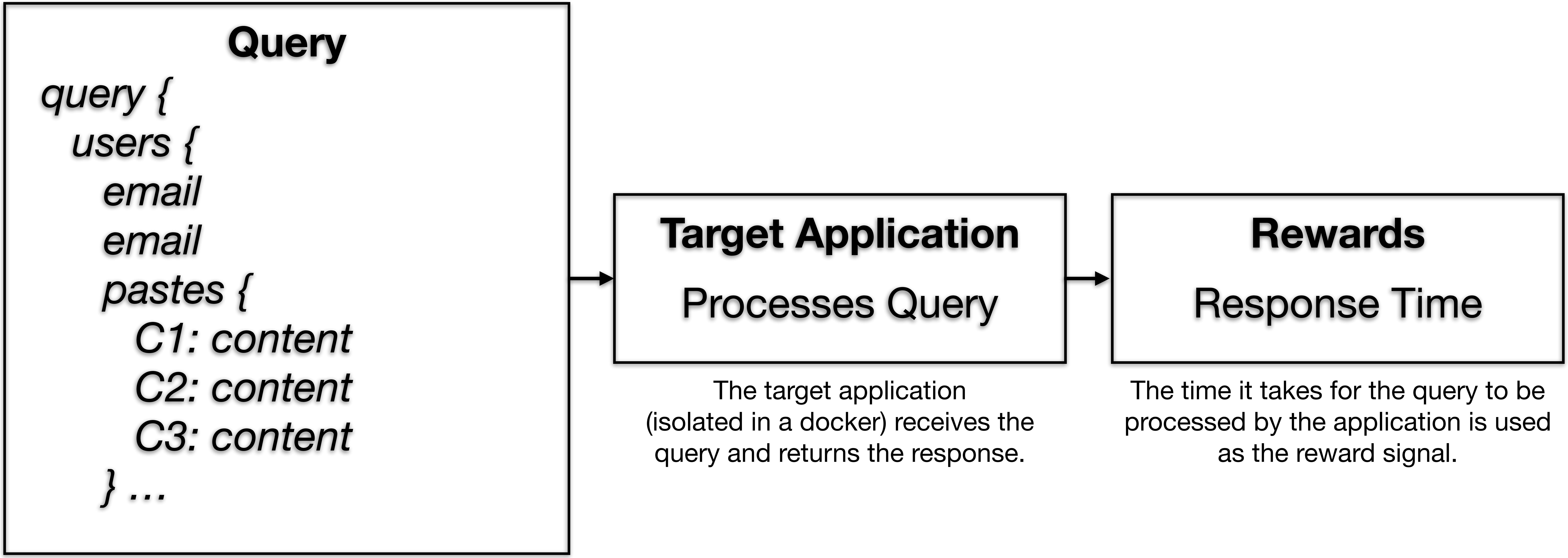
**New Query**

```
query {
  users { Depth = 1 & Height = 1
    email | Height = 2
    email |
    pastes { Depth = 2 & Height = 1
      C1: content | Height = 3
      C2: content |
      C3: content |
    } …
```

New query produced by applying action to the current state

# Rewards

## Environment



**Query**

```
query {
  users {
    email
    email
    pastes {
      C1: content
      C2: content
      C3: content
    } …
```

The query produced by performing the action in the last slide.

**Target Application**

Processes Query

The target application (isolated in a docker) receives the query and returns the response.

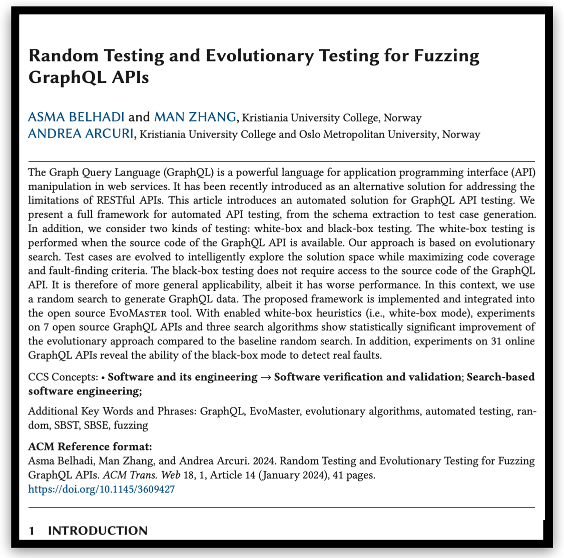
**Rewards**

Response Time

The time it takes for the query to be processed by the application is used as the reward signal.

# Evaluation

## Experimental Settings



**Random Testing and Evolutionary Testing for Fuzzing GraphQL APIs**

ASMA BELHADI and MAN ZHANG, Kristiania University College, Norway
ANDREA ARCURI, Kristiania University College and Oslo Metropolitan University, Norway

The Graph Query Language (GraphQL) is a powerful language for application programming interface (API) manipulation in web services. It has been recently introduced as an alternative solution for addressing the limitations of RESTful APIs. This article introduces an automated solution for GraphQL API testing. We present a full framework for automated API testing, from the schema extraction to test case generation. In addition, we consider two kinds of testing: white-box and black-box testing. The white-box testing is performed when the source code of the GraphQL API is available. Our approach is based on evolutionary search. Test cases are evolved to intelligently explore the solution space while maximizing code coverage and fault-finding criteria. The black-box testing does not require access to the source code of the GraphQL API. It is therefore of more general applicability, albeit it has worse performance. In this context, we use a random search to generate GraphQL data. The proposed framework is implemented and integrated into the open source EvoMaster tool. With enabled white-box heuristics (i.e., white-box mode), experiments on 7 open source GraphQL APIs and three search algorithms show statistically significant improvement of the evolutionary approach compared to the baseline random search. In addition, experiments on 31 online GraphQL APIs reveal the ability of the black-box mode to detect real faults.

CCS Concepts: • **Software and its engineering → Software verification and validation**; **Search-based software engineering;**

Additional Key Words and Phrases: GraphQL, EvoMaster, evolutionary algorithms, automated testing, random, SBST, SBSE, fuzzing

**ACM Reference format:**
Asma Belhadi, Man Zhang, and Andrea Arcuri. 2024. Random Testing and Evolutionary Testing for Fuzzing GraphQL APIs. *ACM Trans. Web* 18, 1, Article 14 (January 2024), 41 pages.
https://doi.org/10.1145/3609427

1 INTRODUCTION

***EvoMaster***

State-of-the-art GraphQL Fuzzer

***Random***

*Wendigo* w/
Random Action Selection

***Random-Greedy***

Wendigo-*Random* w/
Greedy State Selection

***PPO***

*Wendigo* w/
PPO Action selection

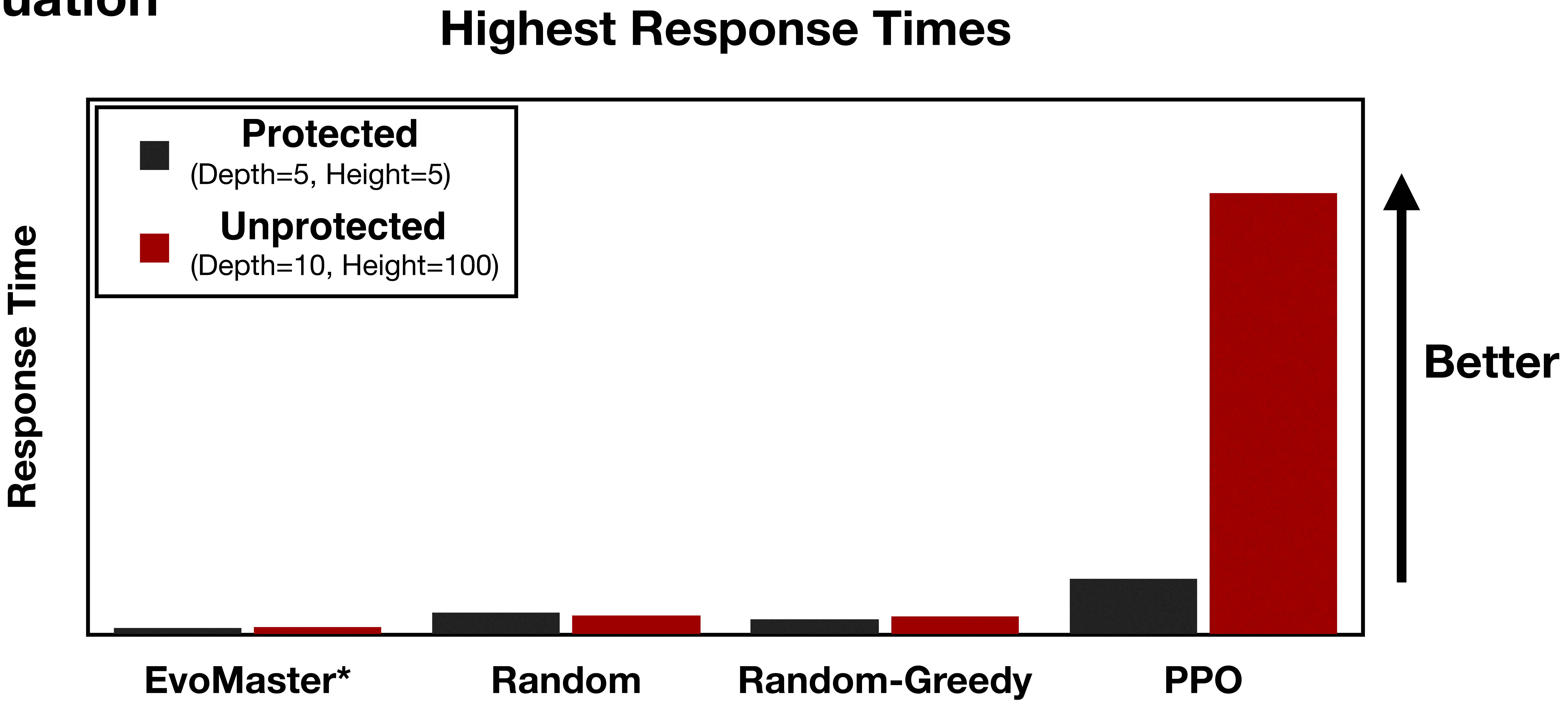
***DVGA***

DVGA Logo Source: https://github.com/dolevf/Damn-Vulnerable-GraphQL-Application

Dice Image Credit: SDXL-Lightning
PPO Logo Source: https://openai.com

# LDoS Query Discovery

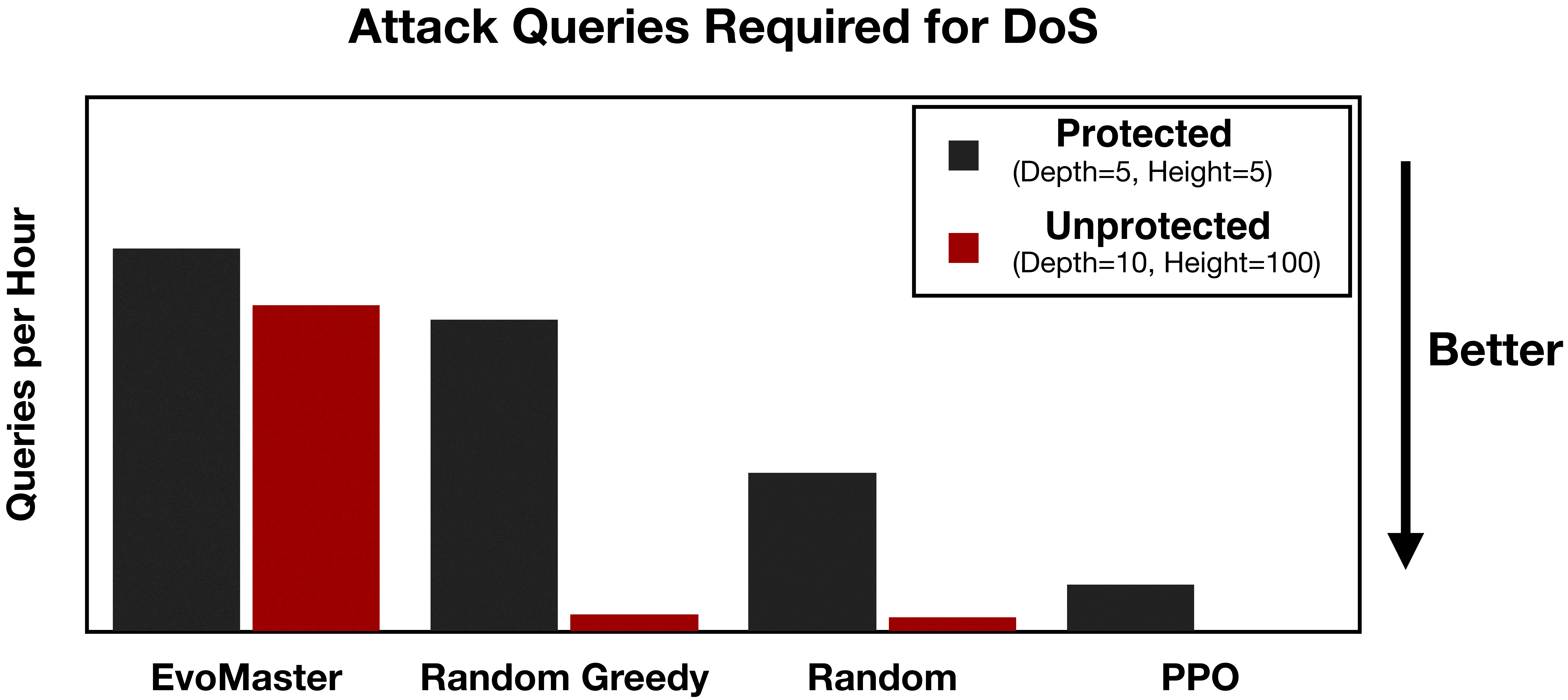
## Evaluation



For reference the PPO results converted to minutes are 3m28s for 208s and 27m30s for 1650s.

# LDoS Attack Impact

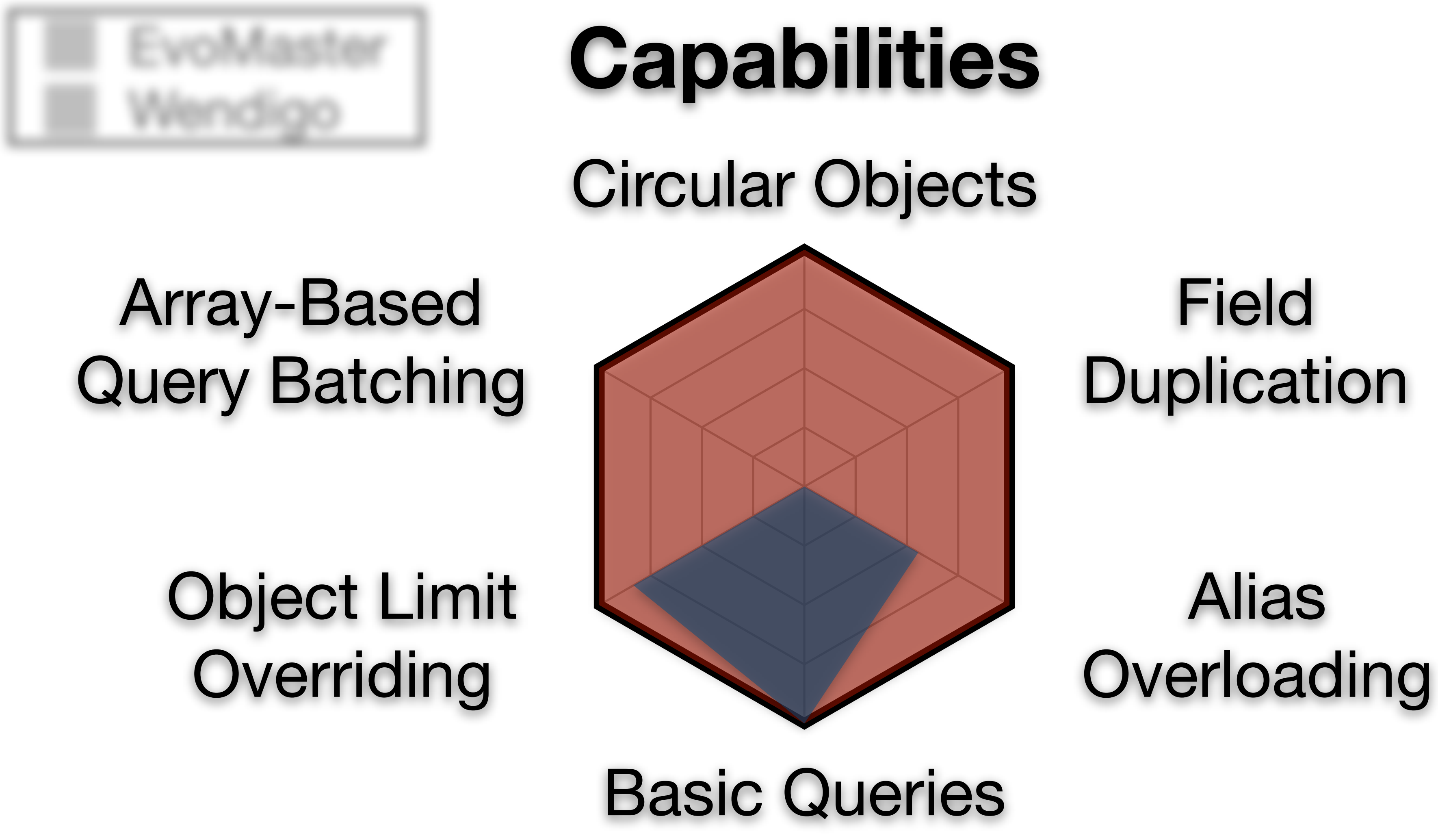
## Evaluation



# Conclusion

## Recap

- DRL approach designed for black-box DoS query discovery.
- Combines multiple DoS attack vectors in GraphQL.
- Outperforms EvoMaster, an existing SOTA fuzzing tool.
- Code has been publicly released.



**Capabilities**

## Future Work

- Extended Capabilities
- Other DRL Approaches
- Evolutionary Baseline
- Open-Source Projects

Supported by

# Thank you

## Wendigo: Deep Reinforcement Learning for Denial-of-Service Query Discovery in GraphQL

KING'S College LONDON

Code

Paper

Website

The Alan Turing Institute

aicd

Supported by

[dstl]
The Science Inside

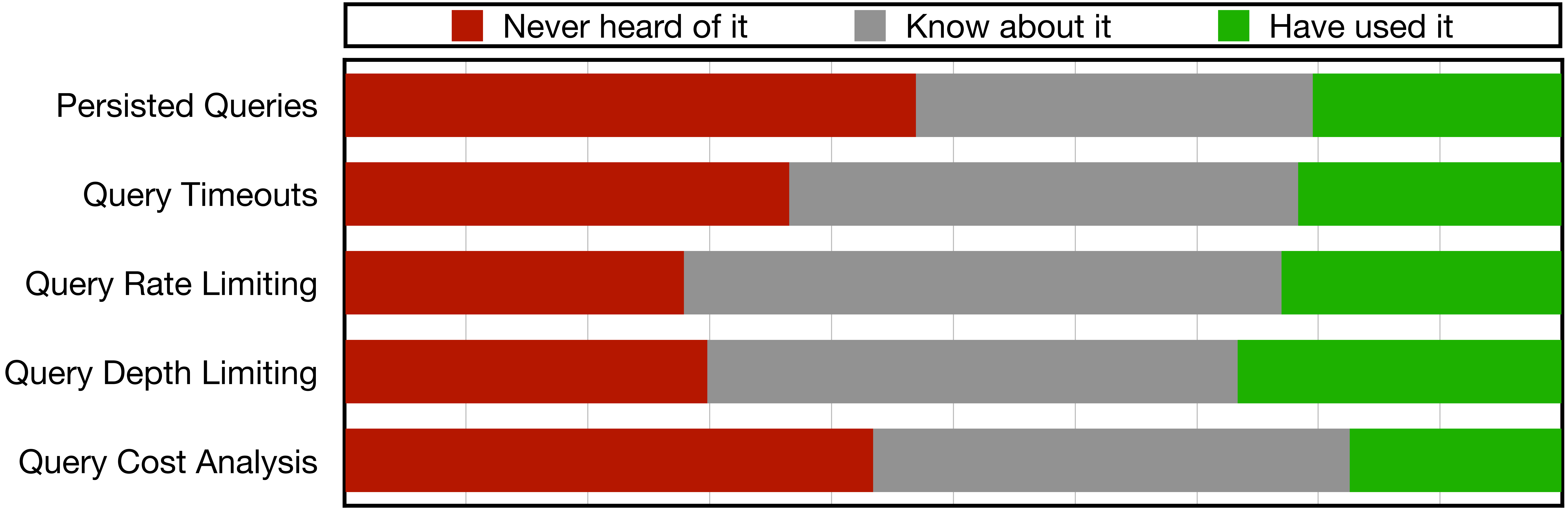
**Shae McFadden**[1,2], Marcello Maugeri[3,1], Chris Hicks[2], Vasilios Mavroudis[2], Fabio Pierazzi[1]
[1]King's College London, [2]The Alan Turing Institute, [3]University of Catania

SICILIAE STVDIVM GENERALE · 1434 ·

Università di Catania

# Usage of DoS Defences

## GraphQL



Results from a 2022 survey of over two-thousand developers.

Source: https://2022.stateofgraphql.com/

# DoS Attack Vectors

## GraphQL

### Field Duplication

```
query {
  pastes {
    content
    content
    ...
    content
    content
  }
}
```

Duplicate to cause the server to repeat computation.

### Alias Overloading

```
query {
  pastes {
    C1: content
    C2: content
    ...
    C100: content
    C101: content
  }
}
```

Alternative form of duplication under new return names.

### Array-Based Query Batching

```
[ query {
      pastes {
        content
      }
  }, query {
        pastes {
          content
        }
      }
]
```

Duplication of entire queries in a singular request.

Attack Vector Source: N. Aleks, et al. Black Hat GraphQL: Attacking Next Generation APIs.

# DoS Attack Vectors

## GraphQL

### Object Limit Overriding

```
query {
  pastes (limit: 1000) {
    content
  }
}
```

Increase pagination arguments to increase the number of objects to be returned.

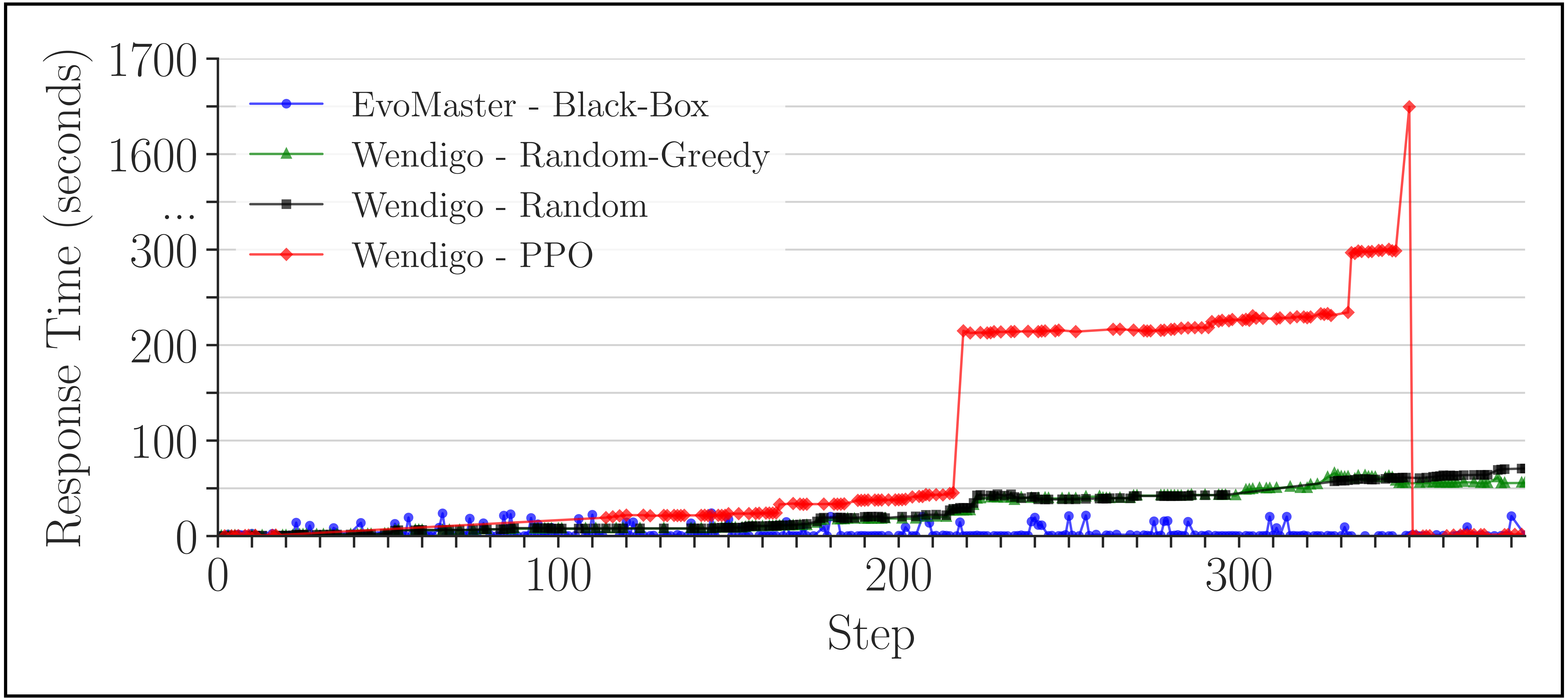
### Circular Objects

```
query {
  pastes {
    owner {
      pastes {
        content
      }
    }
  }
}
```

Recursive cycle of object references in query resulting in a recursive expansion when generating response.

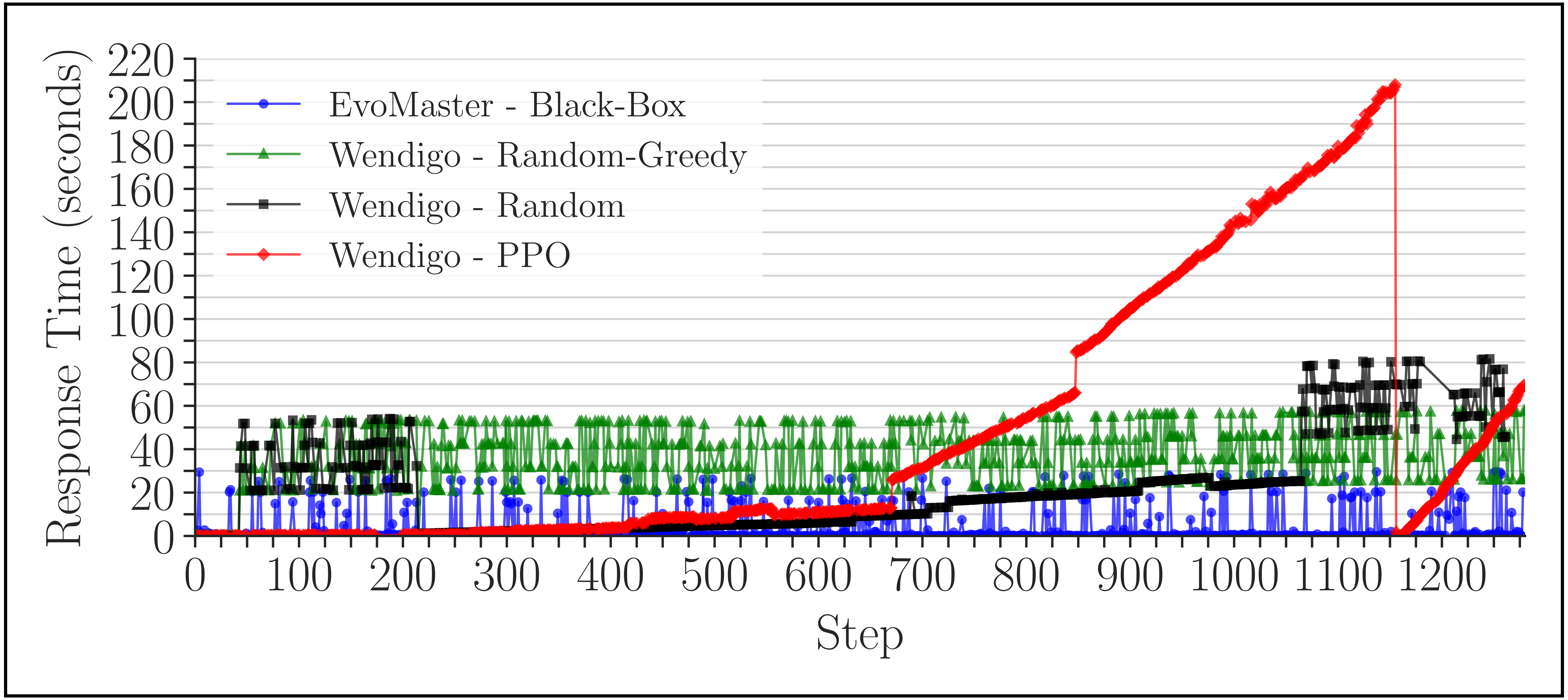
Attack Vector Source: N. Aleks, et al. Black Hat GraphQL: Attacking Next Generation APIs.

# Unprotected Setting



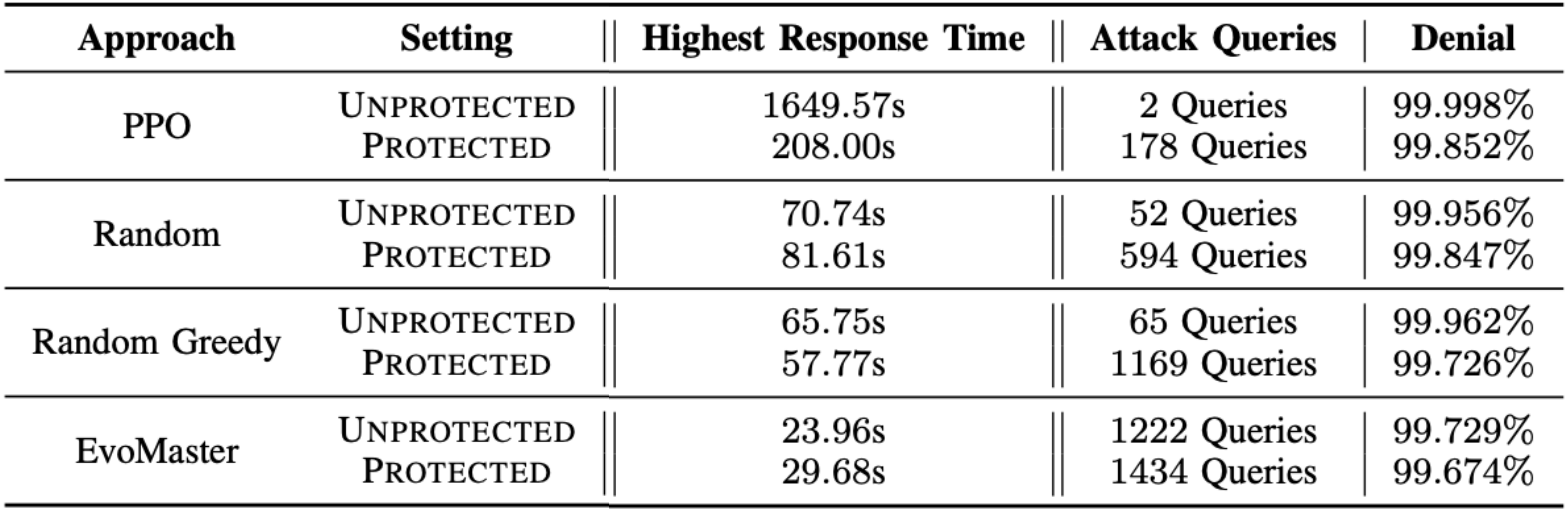
Evaluates an application with no DoS mitigations in place.
*Max_Depth*=10, *Max_Height*=100, *Multiplier*=10

# Protected Setting



Evaluates an application with basic DoS mitigations in place.
*Max_Depth*=5, *Max_Height*=5, *Multiplier*=1

# LDoS Attack Results



| Approach | Setting | Highest Response Time | Attack Queries | Denial |
|---|---|---|---|---|
| PPO | Unprotected | 1649.57s | 2 Queries | 99.998% |
| | Protected | 208.00s | 178 Queries | 99.852% |
| Random | Unprotected | 70.74s | 52 Queries | 99.956% |
| | Protected | 81.61s | 594 Queries | 99.847% |
| Random Greedy | Unprotected | 65.75s | 65 Queries | 99.962% |
| | Protected | 57.77s | 1169 Queries | 99.726% |
| EvoMaster | Unprotected | 23.96s | 1222 Queries | 99.729% |
| | Protected | 29.68s | 1434 Queries | 99.674% |

Determine the number of queries required to perform a DoS attack utilizing the percentage of denied benign user's query period for calibration.