Eywa : Automating Model-Based Testing using LLMs

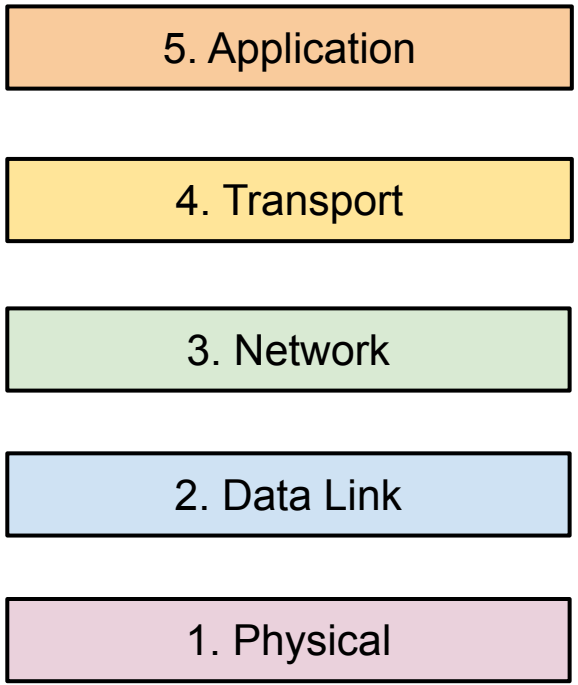
Rajdeep Mondal, Rathin Singha, Todd Millstein, George Varghese, Ryan Beckett, Siva Kesava Reddy Kakarla

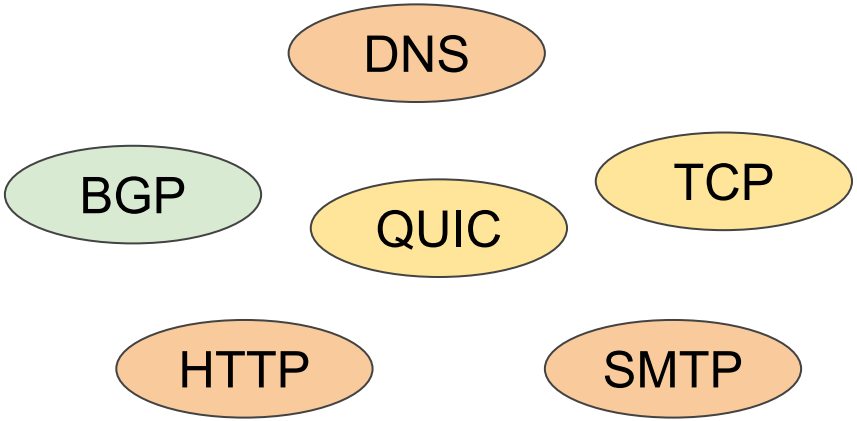
UCLA

Microsoft

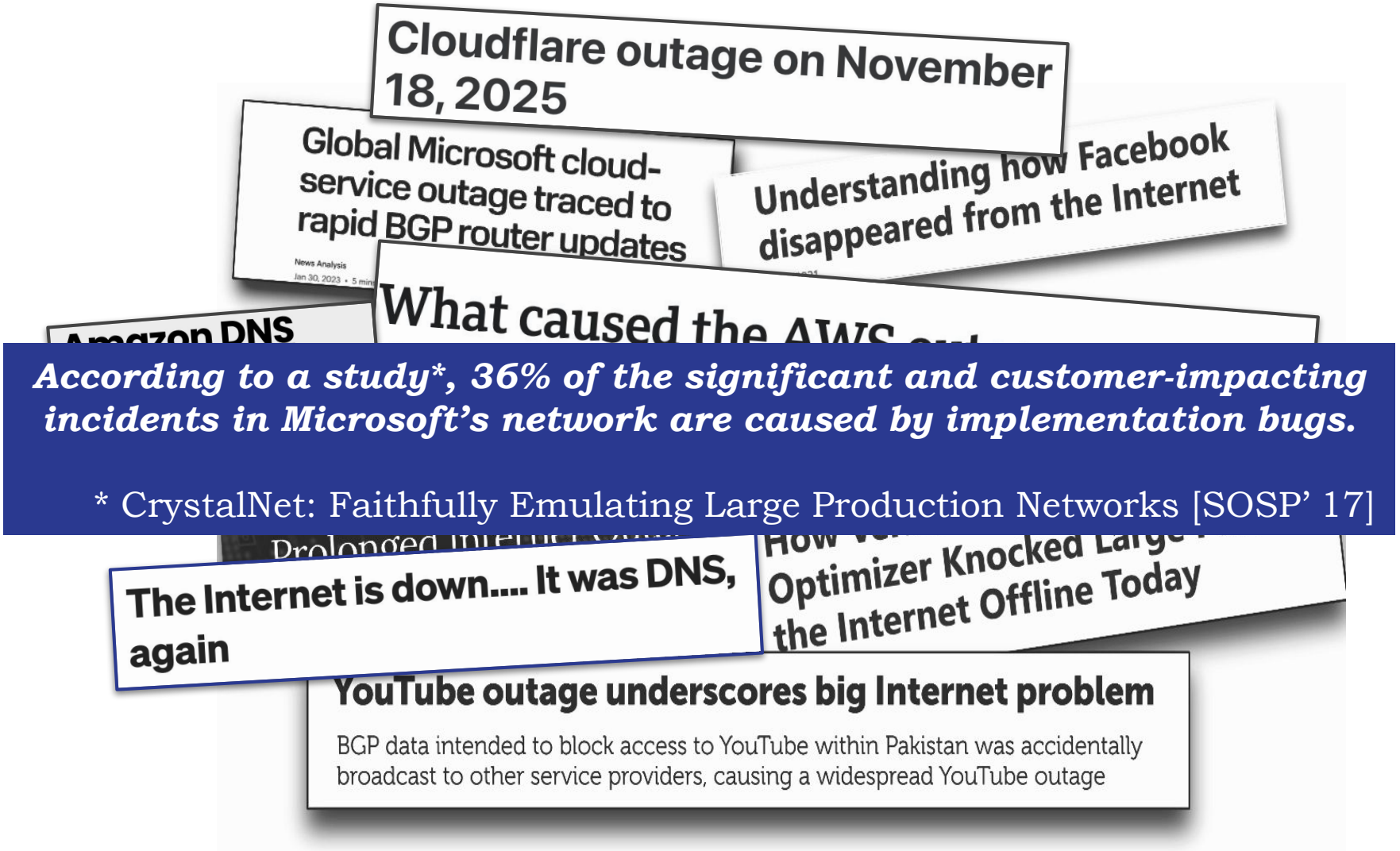
Github link: https://github.com/microsoft/Model_Based_Testing_Using_LLMs

Why Protocol Testing Matters?





- Protocols are the backbone of internet
- Implementations can be buggy
- Consequences: Outages, security vulnerabilities

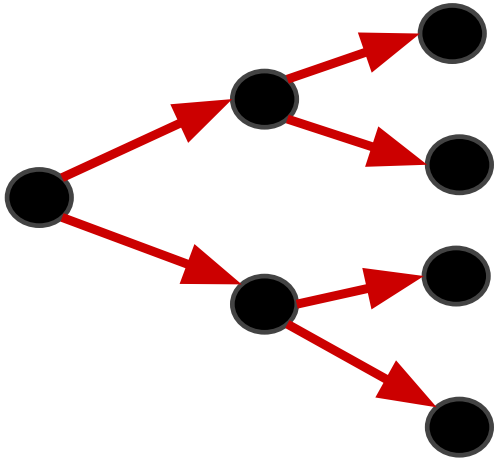


According to a study*, 36% of the significant and customer-impacting incidents in Microsoft's network are caused by implementation bugs.

* CrystalNet: Faithfully Emulating Large Production Networks [SOSP' 17]

Model-based Testing

- Small behavioral model created from spec/RFCs
- Explore the model to derive test cases

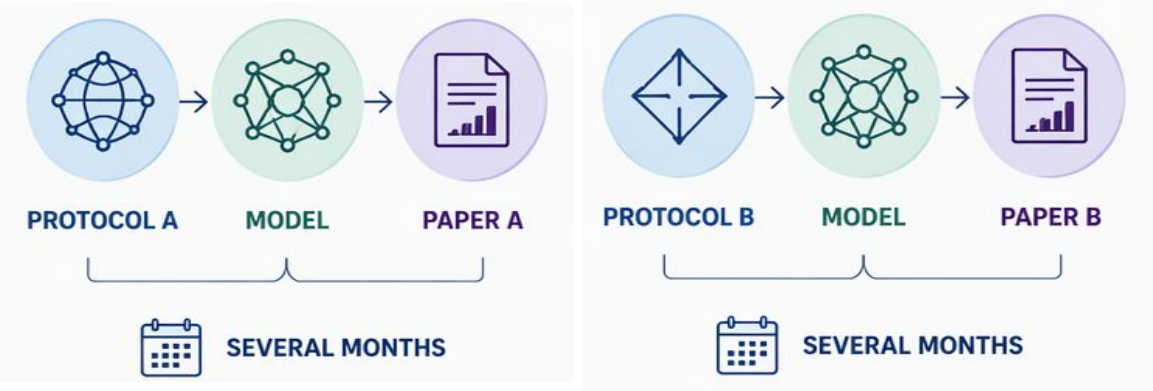


Effective in capturing behavioral discrepancies and critical semantic bugs

Formal specification and testing of QUIC [SIGCOMM' 19]	**27 bugs**
SCALE: Automatically Finding RFC Compliance Bugs in DNS Nameservers [NSDI' 22]	**30 bugs**
MESSI: Behavioral Testing of BGP Implementations [NSDI' 24]	**22 bugs**

Worth the effort?

Months of effort per protocol—one protocol, one research paper.



- Models must be written by hand
- Difficult to maintain for protocol updates
- Challenging for complex protocols like BGP

Our Goal

Remove the manual model building bottleneck in model-based testing

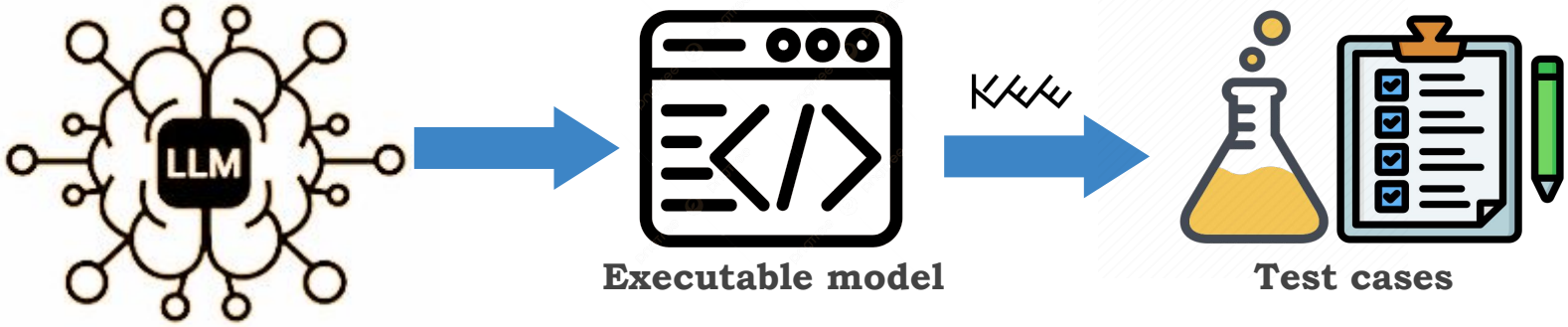
Eywa: The first approach that allows users to build network protocol models and generate test cases quickly and efficiently

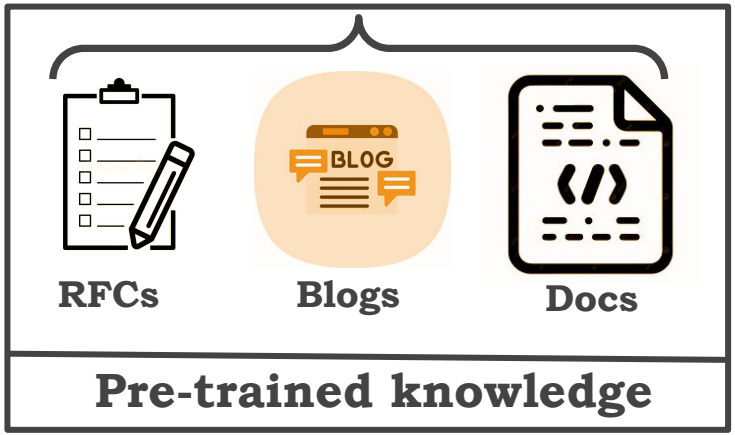
Key Results

Model building takes minutes—with minimal user effort.

- 3 network protocols tested - DNS, BGP and SMTP
- Models for previously unexplored components of BGP - Confederations and Route Reflection
- 110K+ test cases generated
- 33 unique bugs detected across 16 different implementations

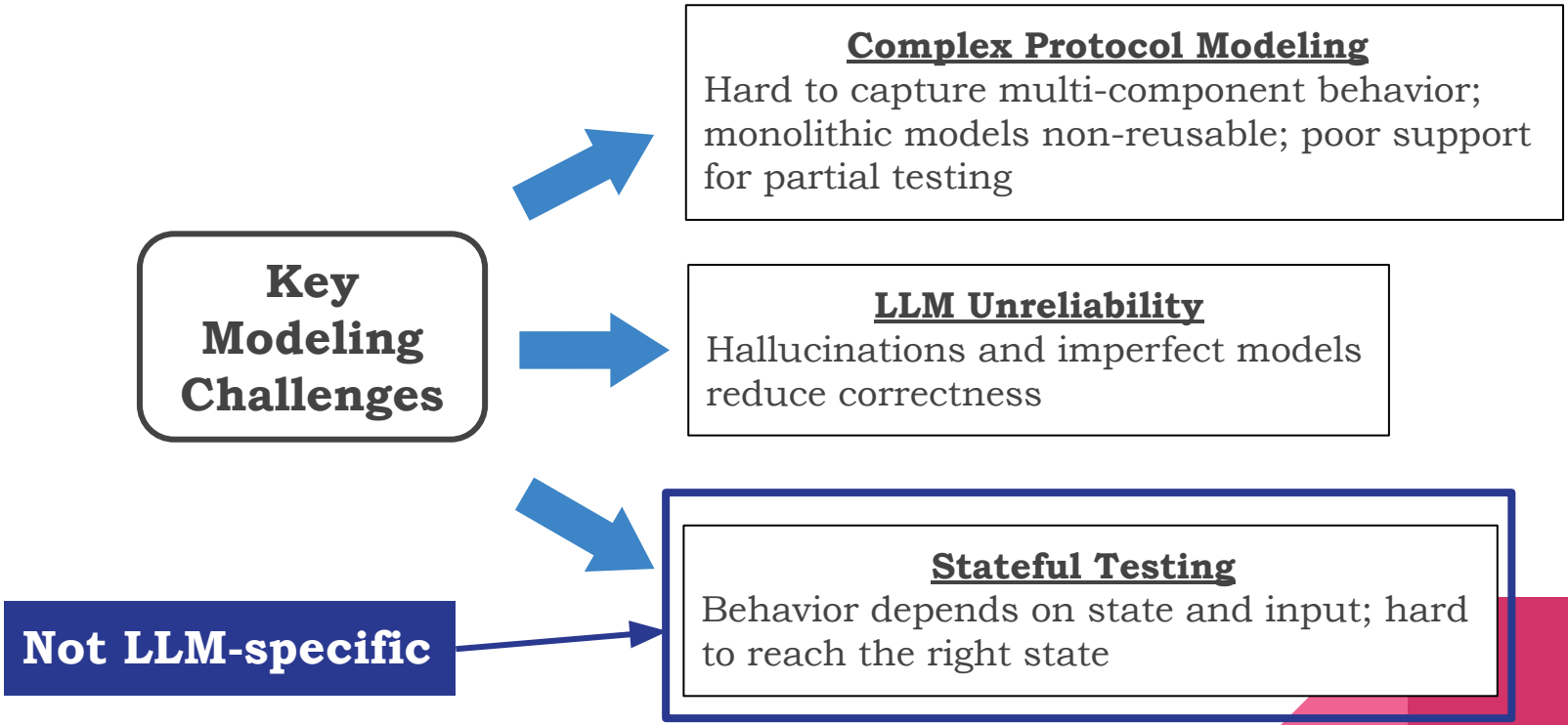
A first idea…





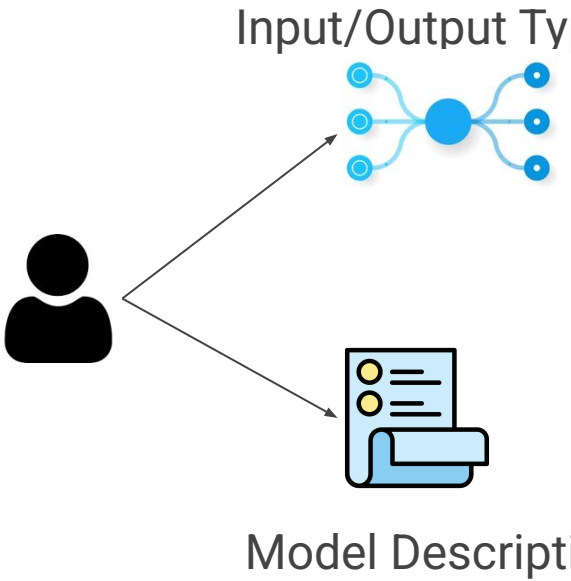
Simple abstract English prompt specifying which protocol to test

Not that simple …



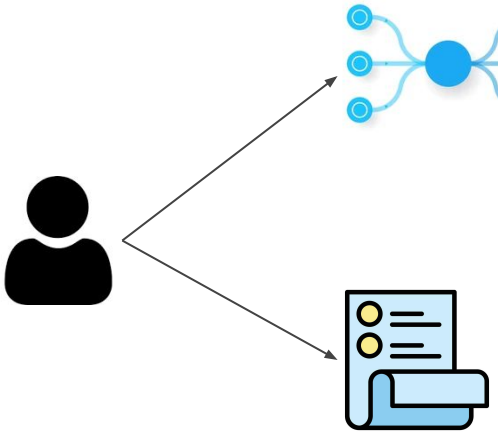
Challenge 1: Modeling complex protocols

Represent protocol components as function modules



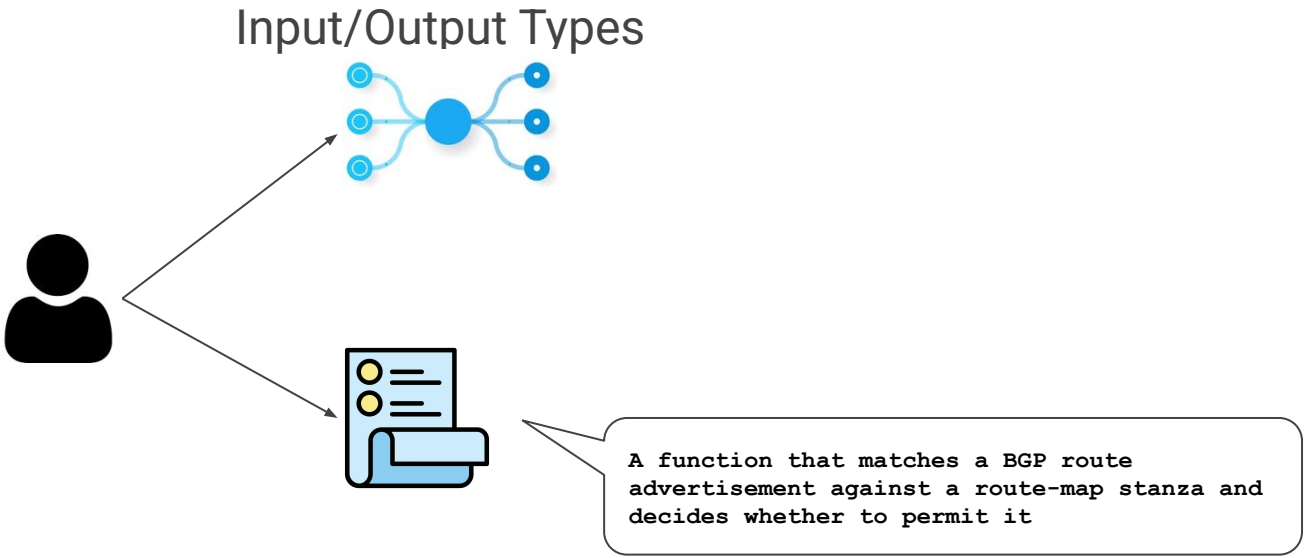
Challenge 1: Modeling complex protocols

```
route = eywa.Struct("Route",
                    ipAddr = eywa.Int(32),
                    prefixMask = eywa.Int(32),
                    localPref = eywa.Int(32))
rmap = eywa.Struct("RMapStanza",
                    localPref = eywa.Int(32),
                    permit = eywa.Bool())
```

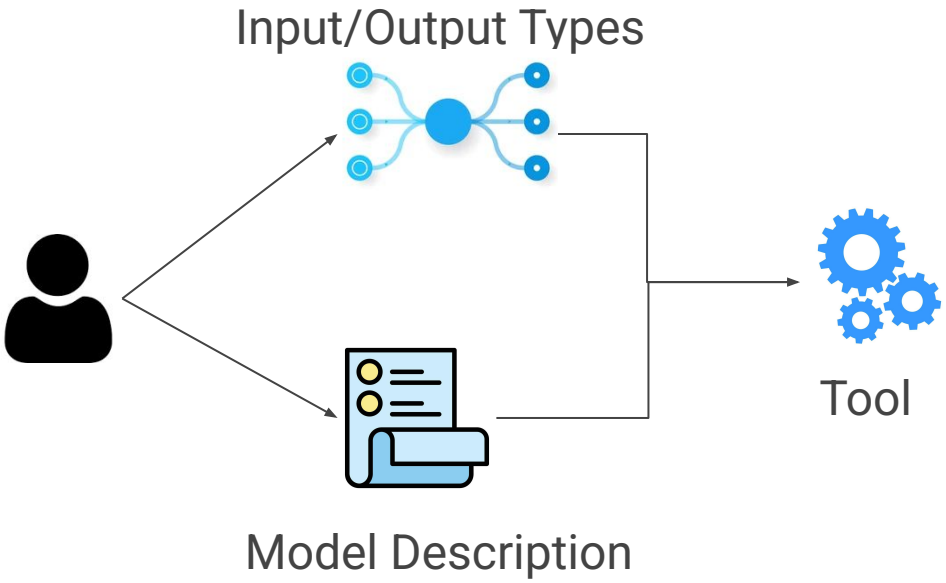


Model Description

Challenge 1: Modeling complex protocols



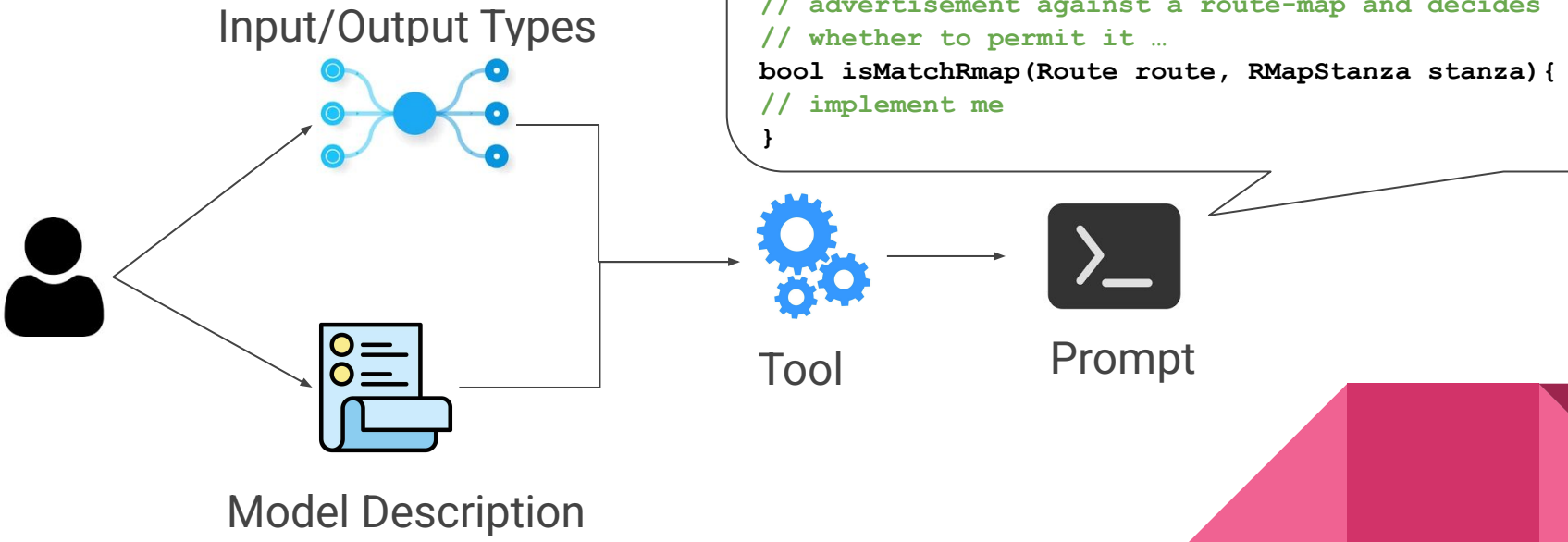
Challenge 1: Modeling complex protocols



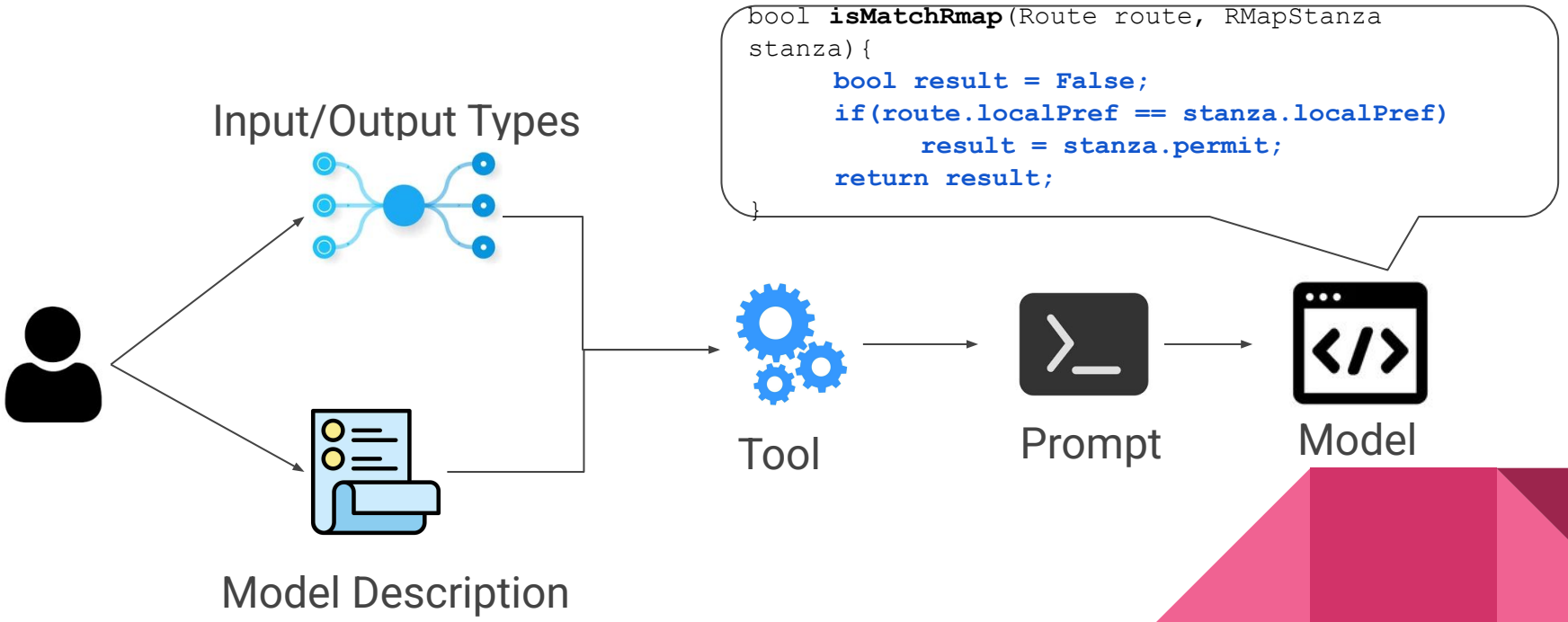
Challenge 1: Modeling complex protocols

```
typedef struct Route {...}
typedef struct RMapStanza {...}

// A function that matches a BGP route
// advertisement against a route-map and decides
// whether to permit it …
bool isMatchRmap(Route route, RMapStanza stanza){
// implement me
}
```

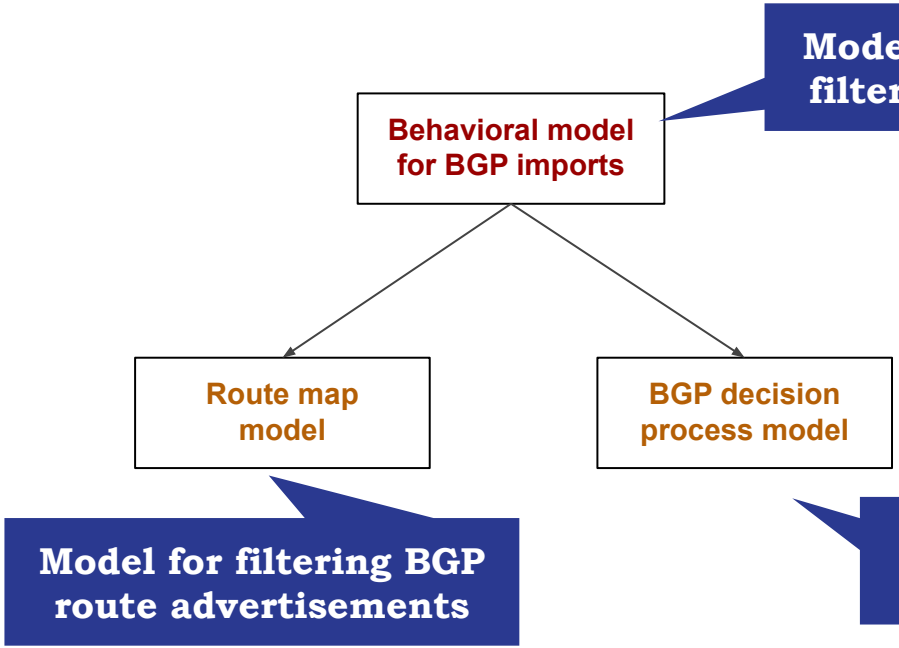


Challenge 1: Modeling complex protocols



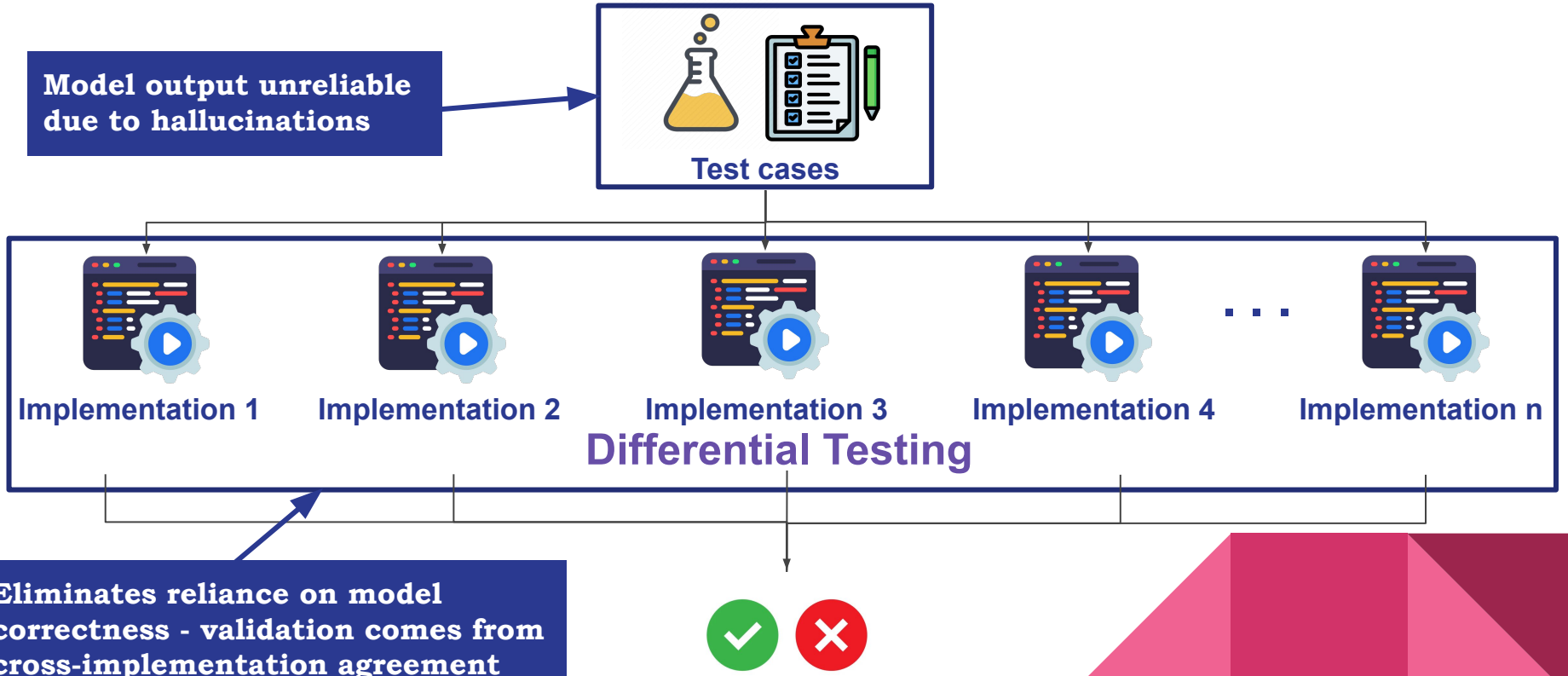
Challenge 1: Modeling complex protocols

Compose multiple modules using dependency graphs

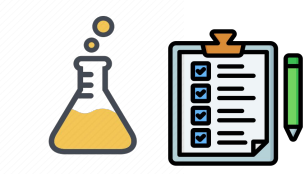


- Separate LLM invocations for each module
- User provided dependency graph → composes into a complete model

Challenge 2: Dealing with imperfect models



Challenge 2: Dealing with imperfect models



Test cases

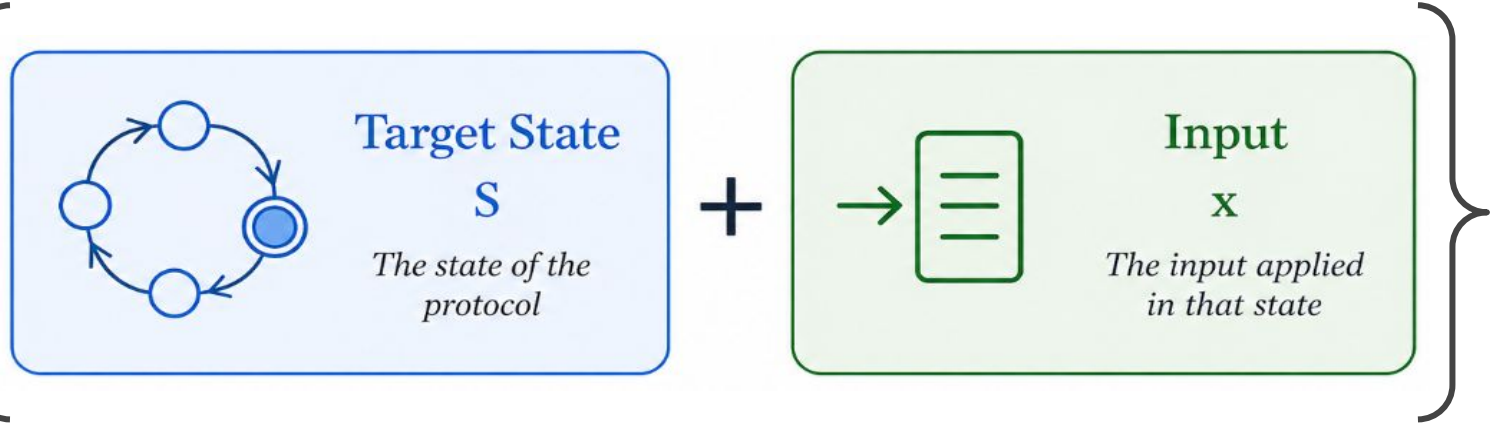
Increase test case diversity by invoking the LLM multiple times to generate different versions of the same model

Implementation 1 | Implementation 2 | Implementation 3 | Implementation 4 | Implementation n

Differential Testing

Challenge 3: Handling stateful protocols

Test case =



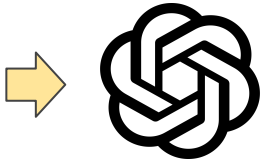
We need to to drive the protocol to the desired state before testing its behavior

Challenge 3: Handling stateful protocols

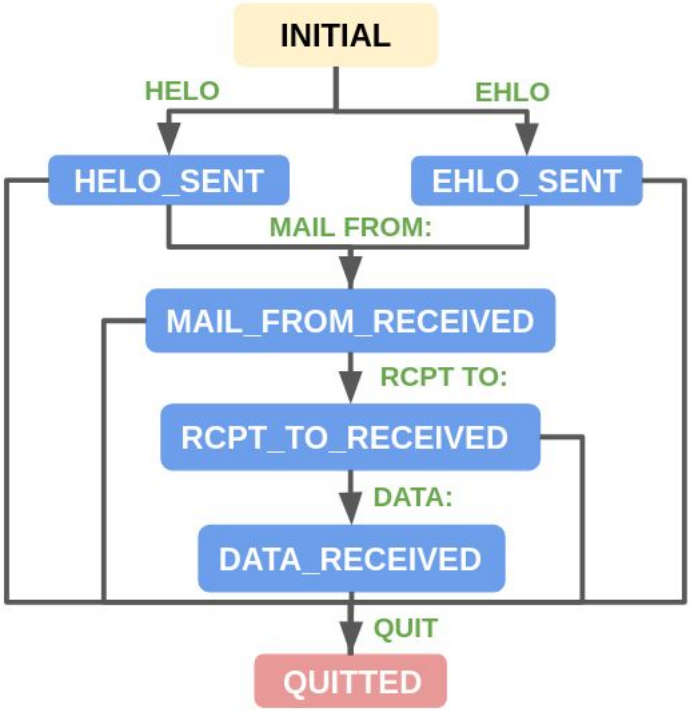
Leverage the LLM-generated model to obtain the state graph!

```
char* smtp_server_response(State state, char* input) {
    char* response = malloc(512 * sizeof(char));
    if(state == INITIAL) {
        if(strcmp(input, "HELO") == 0) {
            strcpy(response, "250 Hello");
            state = HELO_SENT;
        } else if(strcmp(input, "EHLO") == 0) {
            strcpy(response,
            "250-Hello\n250-SIZE 512\n250-8BITMIME\n250 OK")
            state = EHLO_SENT;
        } else {
            strcpy(response, "503 Bad sequence of commands")
        }
    } else if(state == HELO_SENT || state == EHLO_SENT) {
        if(strncmp(input, "MAIL FROM:", 10) == 0) {
            strcpy(response, "250 OK");
            state = MAIL_FROM_RECEIVED;
        } else {
            strcpy(response, "503 Bad sequence of commands")
        }
    } else if(state == MAIL_FROM_RECEIVED) {
        if(strncmp(input, "RCPT TO:", 8) == 0) {
            strcpy(response, "250 OK");
            state = RCPT_TO_RECEIVED;
        } else {
            strcpy(response, "503 Bad sequence of commands")
        }
    } else if(state == RCPT_TO_RECEIVED) {
        if(strcmp(input, "DATA") == 0) {
            strcpy(response,
            "354 End data with <CR><LF>.<CR><LF>");
            state = DATA_RECEIVED;
        } else {
            strcpy(response, "503 Bad sequence of commands")
        }
    } else if(state == DATA_RECEIVED) {
        if(strcmp(input, ".") == 0) {
            strcpy(response, "250 OK");
            // Reset state to initial after receiving data
            state = INITIAL;
        } else {
            // No specific response for data content,
            // return empty string
            strcpy(response, "");
        }
    } else if(state == QUITTED) {
        strcpy(response, "221 Bye");
```

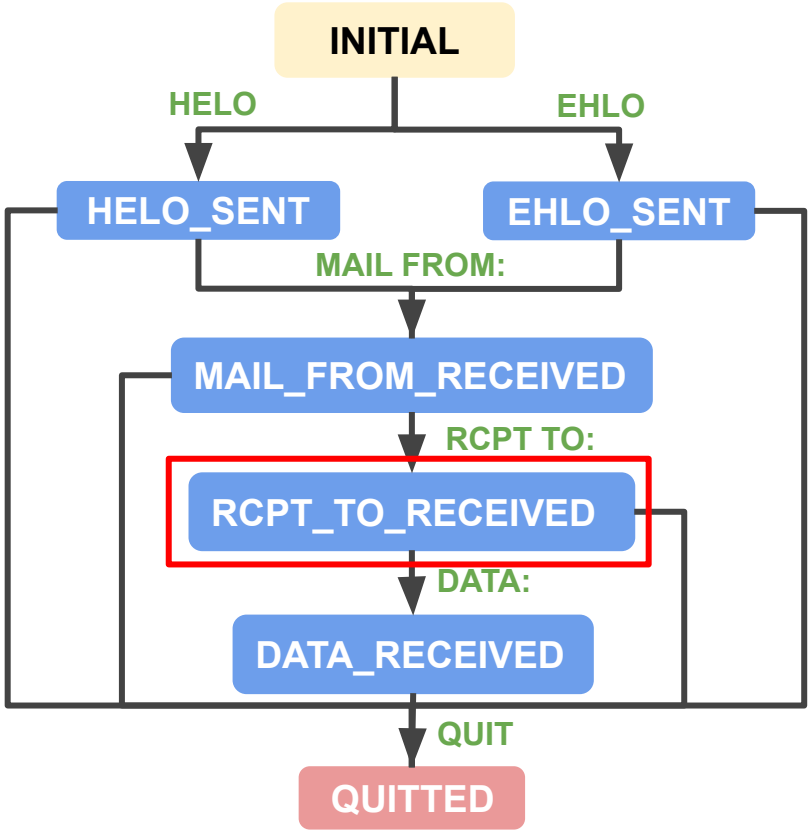
Model for SMTP server



LLM

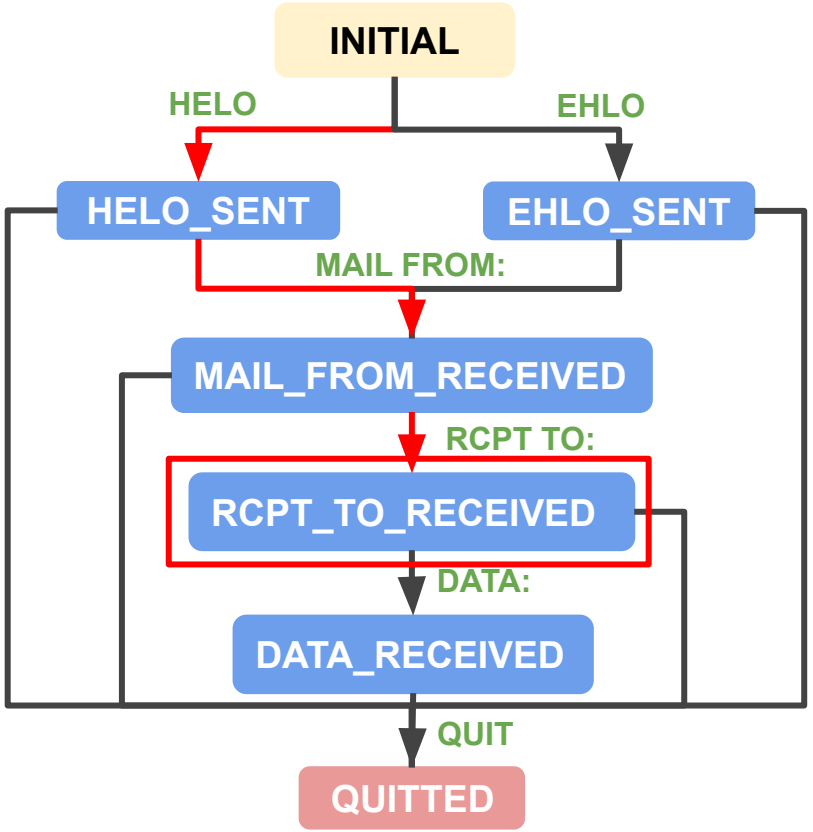


Challenge 3: Handling stateful protocols



Test Case `(RCPT_TO_RECEIVED, "DATA")`

Challenge 3: Handling stateful protocols



Test Case `(RCPT_TO_RECEIVED, "DATA")`

Input Sequence to achieve target state from initial state

`["HELO", "MAIL_FROM", "RCPT_TO"]`

Complete Test Case:
`["HELO", "MAIL_FROM", "RCPT_TO", "DATA"]`

Bug #1 (COREDNS Out-of-zone Response)

Query: <bar.test., A>

Response: NXDOMAIN
Authority Section:
test.test. SOA ....

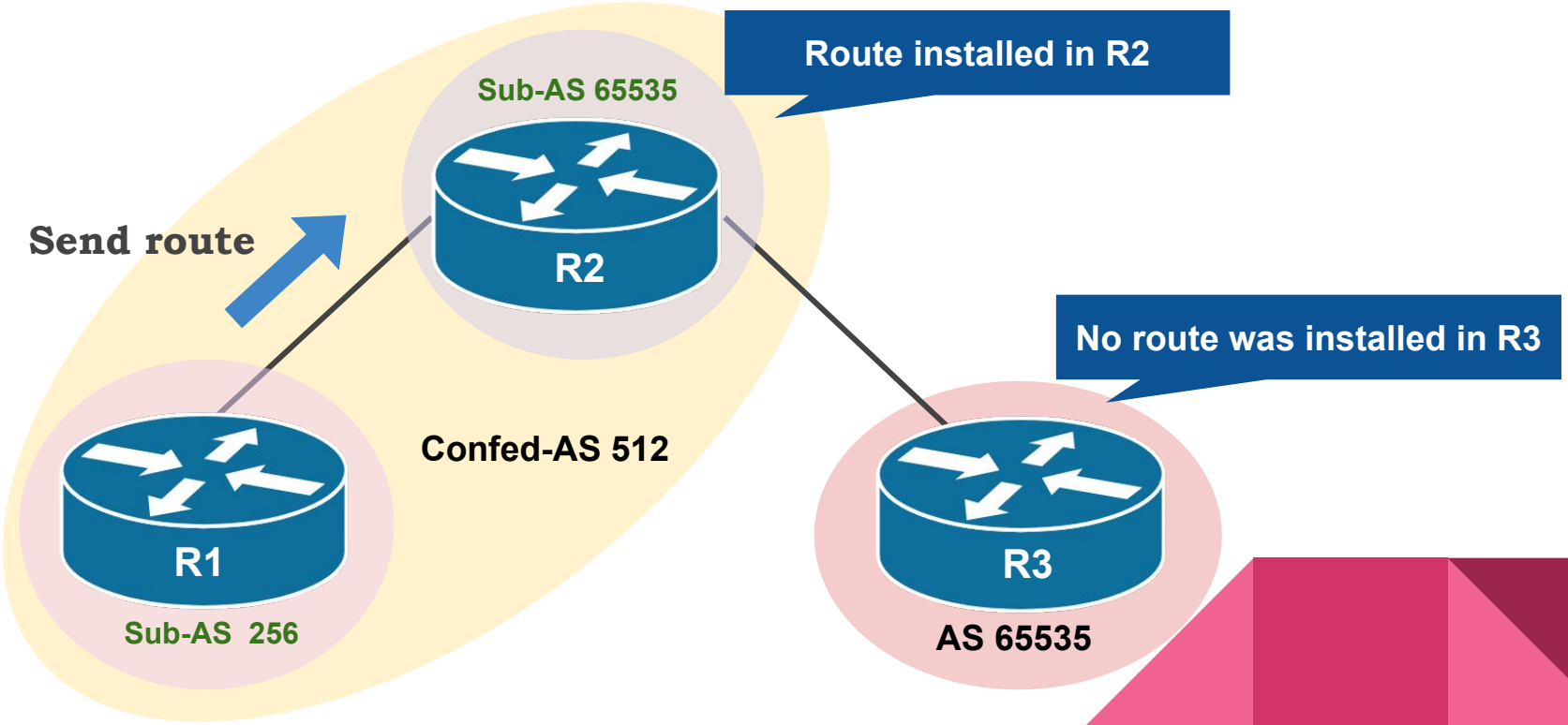
Out-of-zone record

Zone File

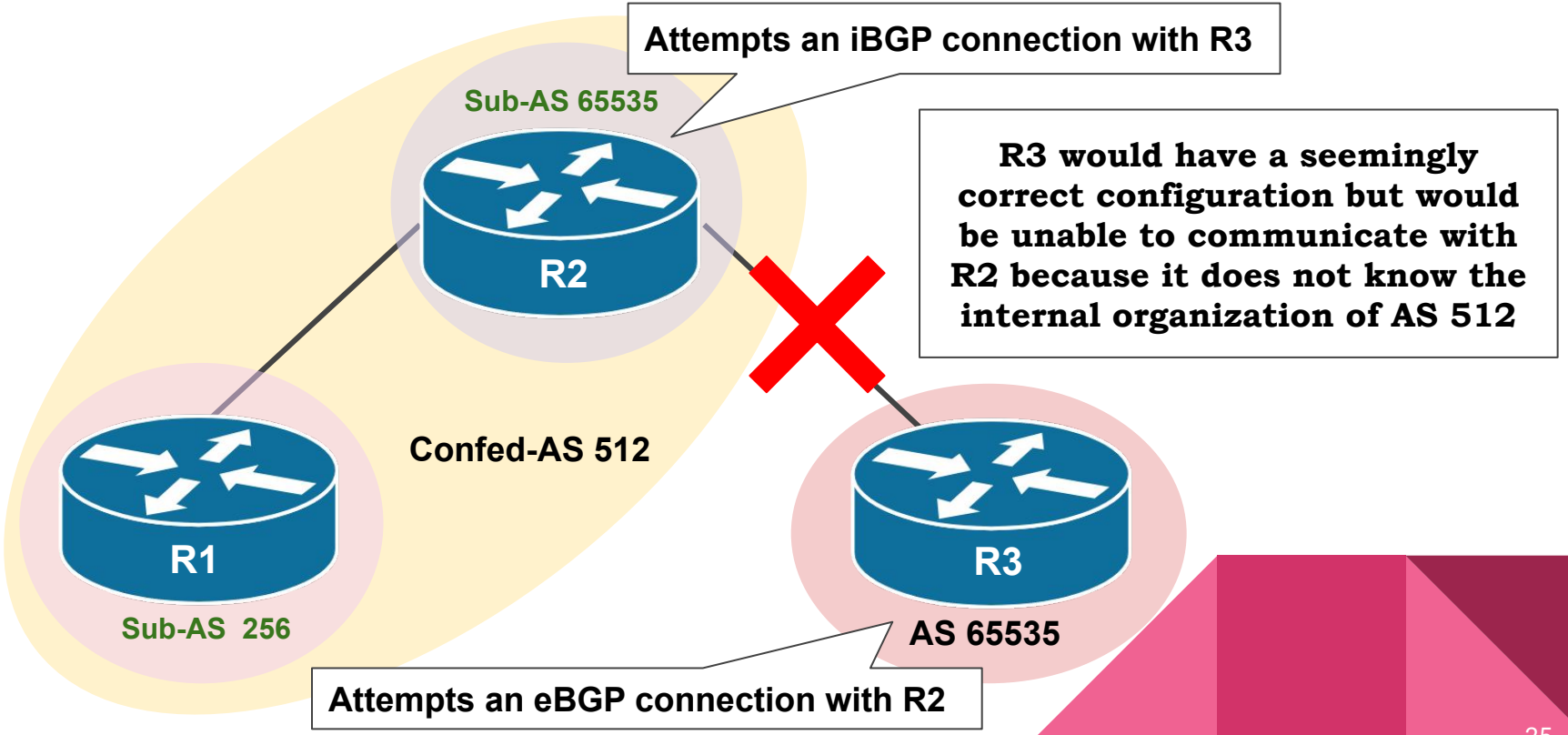
Domain Name	Type	Data
test	SOA	…
test	NS	ns1.outside.edu
foo	A	1.1.1.1

- **Violates DNS trust boundaries**
- **Can cause cache poisoning or performance degradation due to repeated queries**

Bug #2 (BGP Confederations)



Bug #2 (BGP Confederations)



Results

- 3 network protocols tested - DNS, BGP and SMTP
- Models for previously unexplored components of BGP - Confederations and Route Reflection
- 110K+ test cases generated
- 33 unique bugs detected across 16 different implementations

Limitations

- Lacks coverage guarantees: models generated by LLM
- Works best for well-known protocols (LLM prior knowledge)
- User still need some preliminary knowledge of the protocol

Future Work

- Support new protocols by allowing user to provide additional documentation as input

- Address other testing bottlenecks:
  - Setting up test environment
  - Triaging and writing bug reports

Contributions

The first Python library that allows users to efficiently combine LLMs with model-based testing and differential testing

Modularized code synthesis for selective testing of protocol components and their compositions

State graph generation and traversal to generate test cases for stateful protocols

Github link: https://github.com/microsoft/Model_Based_Testing_Using_LLMs