Large Language Models for Validating Network Protocol Parsers

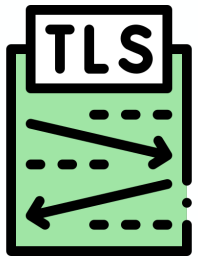
Mingwei Zheng, Danning Xie, Xiangyu Zhang

LangSec 2025

What are Network Protocols?

- Rules that allow devices to communicate

➢ Examples:



Web browsing

IOT

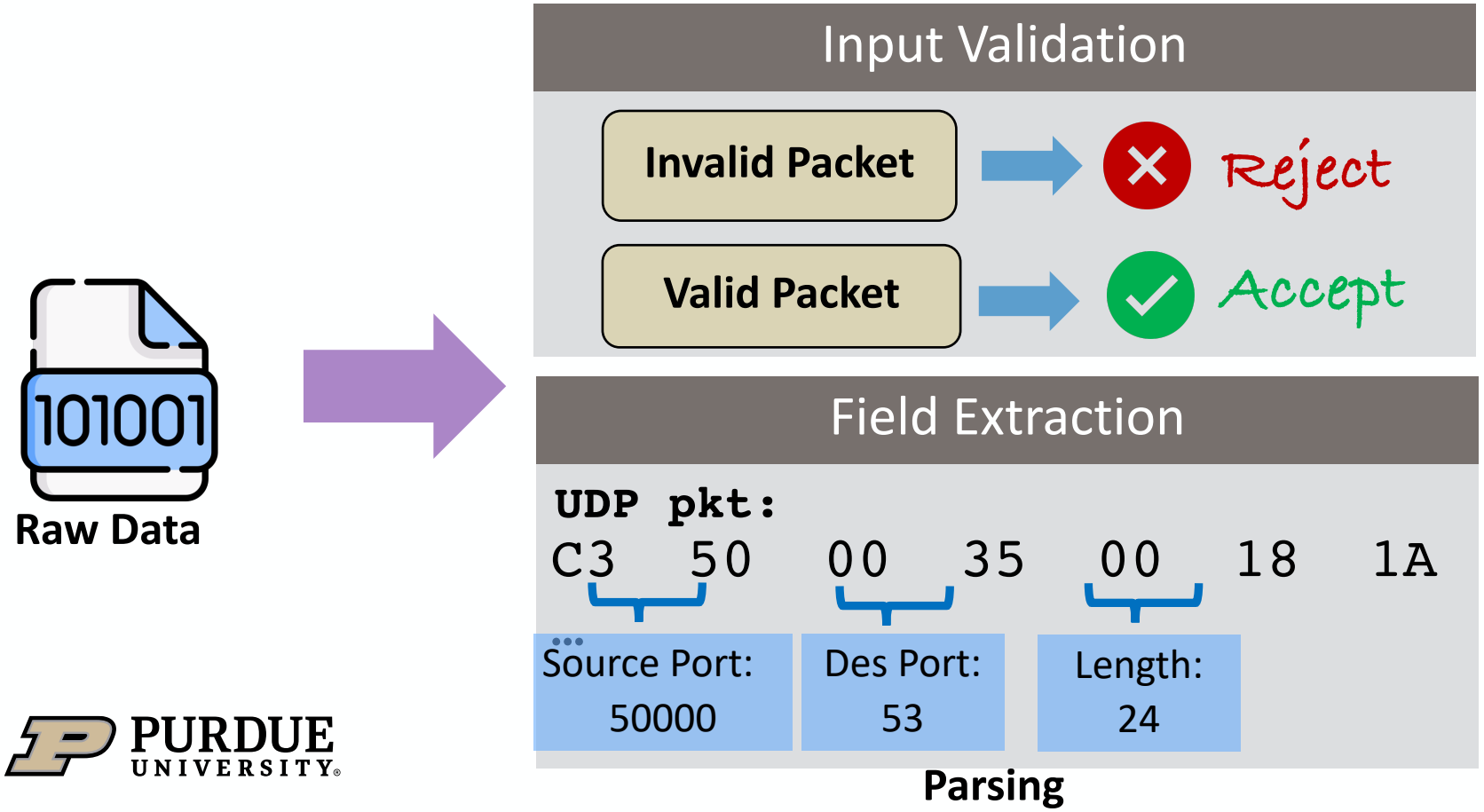
Cloud

PURDUE UNIVERSITY

Network Protocols are used everywhere!

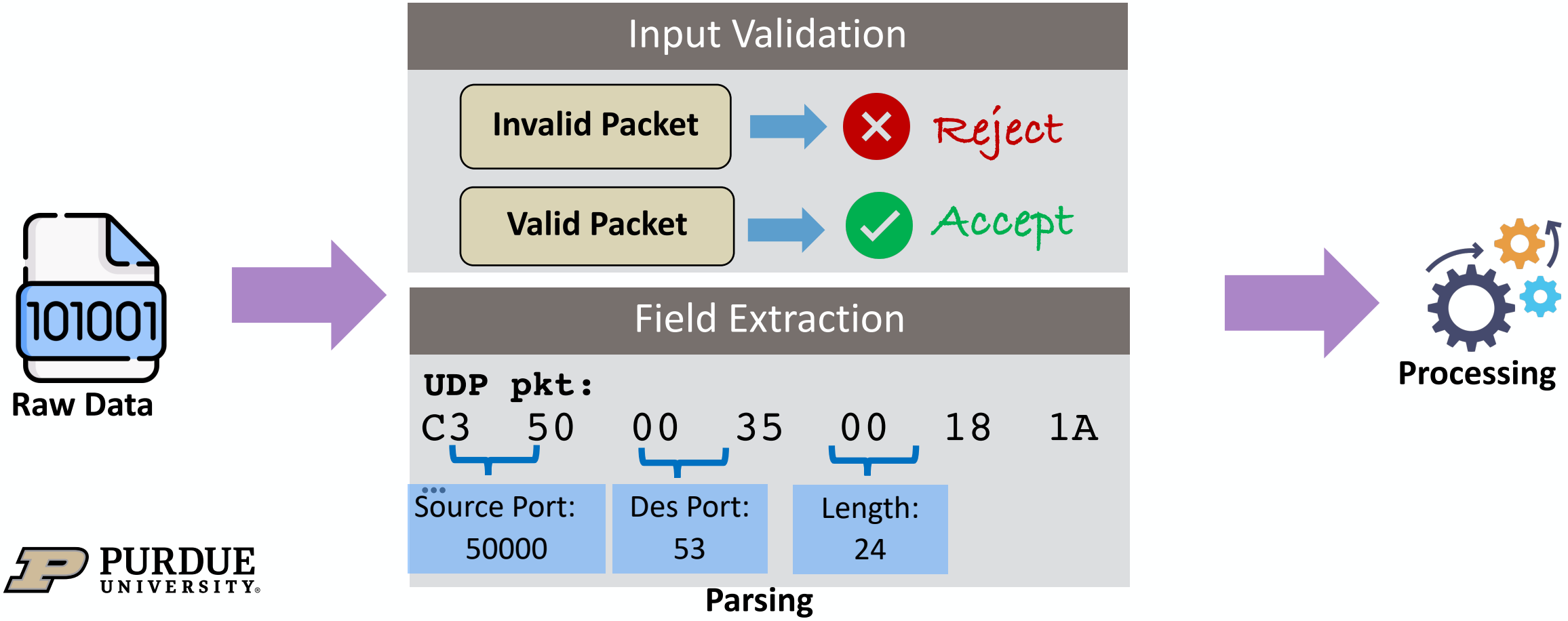
Network Protocol Parsing

- Input Validation: drop malformed packets
- Field Extraction: decode fields from valid packets



Network Protocol Parsing

- Input Validation: drop malformed packets
- Field Extraction: decode fields from valid packets

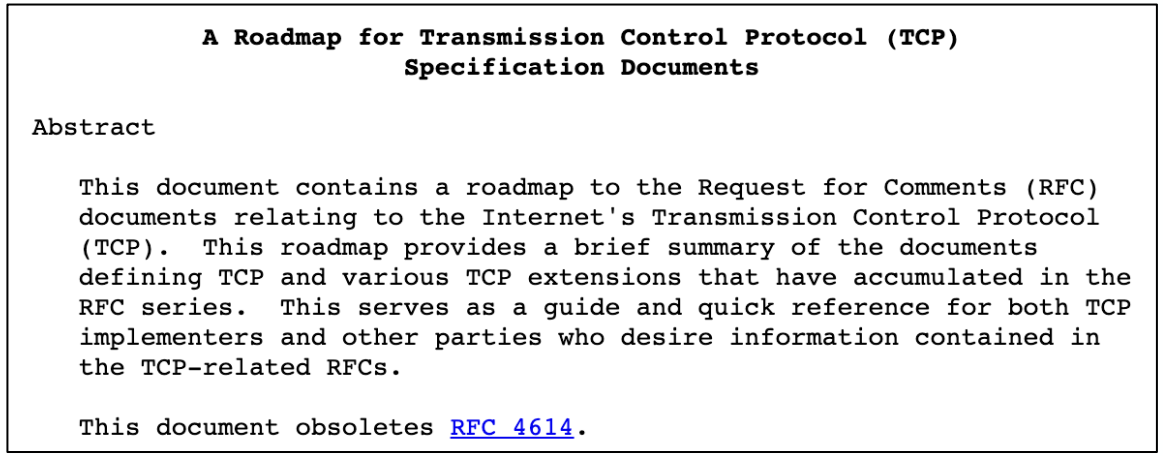


Validating Network Protocol Parsers: Critical but Challenging

- 🐞 **Bug in parsers** → memory corruption, info leakage, DoS attack …

- 📜 Protocol standards (RFCs) in *natural language*; 👨‍💻 parsers in *code*

  → **hard to compare**



A Roadmap for Transmission Control Protocol (TCP) Specification Documents

Abstract

This document contains a roadmap to the Request for Comments (RFC) documents relating to the Internet's Transmission Control Protocol (TCP). This roadmap provides a brief summary of the documents defining TCP and various TCP extensions that have accumulated in the RFC series. This serves as a guide and quick reference for both TCP implementers and other parties who desire information contained in the TCP-related RFCs.

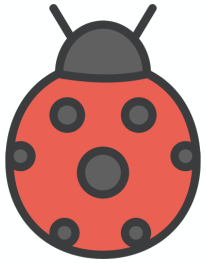
This document obsoletes RFC 4614.

RFC Doc

Parser Implementation

PURDUE UNIVERSITY

Limitations of Existing Techniques



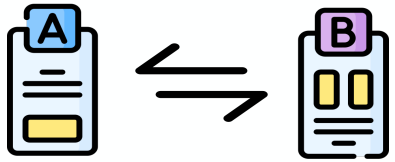
Silent Bugs:
Violate Protocol Rules
Do not cause crashes

Fuzzing

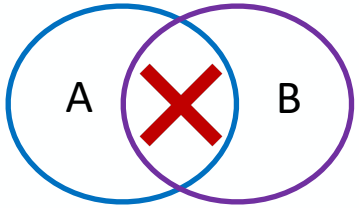
Bug missed

Model Checker

No formal spec



Differential Analysis

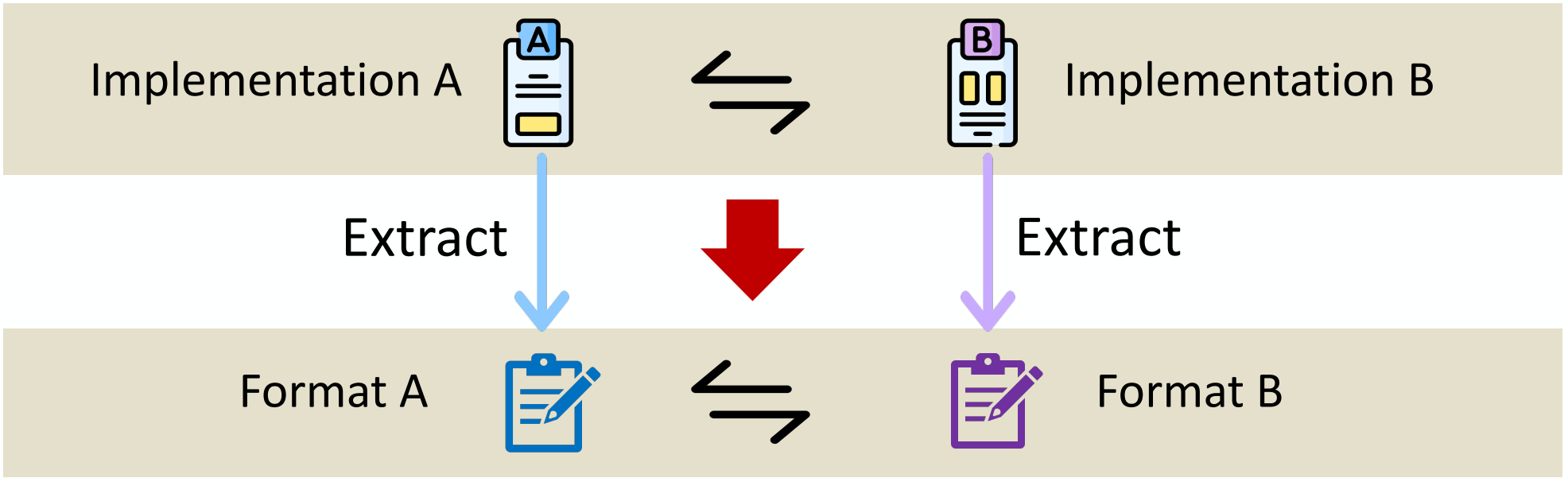


Miss common semantic bugs

PURDUE UNIVERSITY

Motivation

- ParDiff extracts and compares formats from two implementations [1]

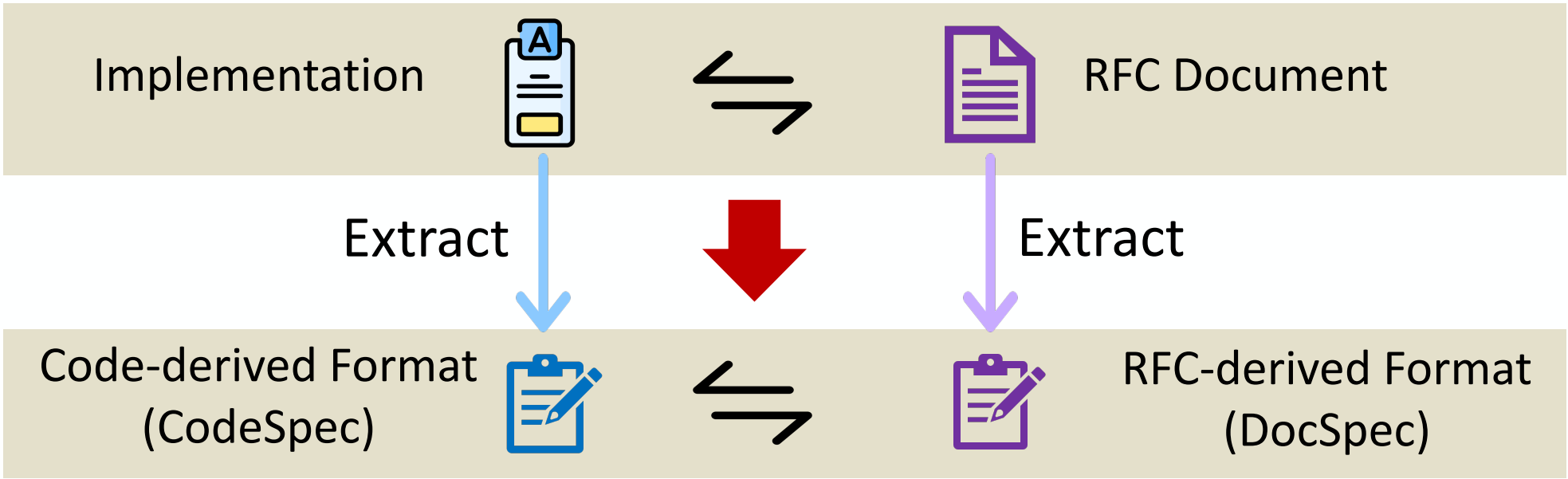


Can we also lift format from RFC documents?

[1] ParDiff: Practical Static Differential Analysis of Network Protocol Parsers (OOPSLA '24)

Our Solution: *Doc. vs. Impl. => Format Spec Comparison*

- Compare **formats** derived from the ***parser code*** and the ***official RFC***

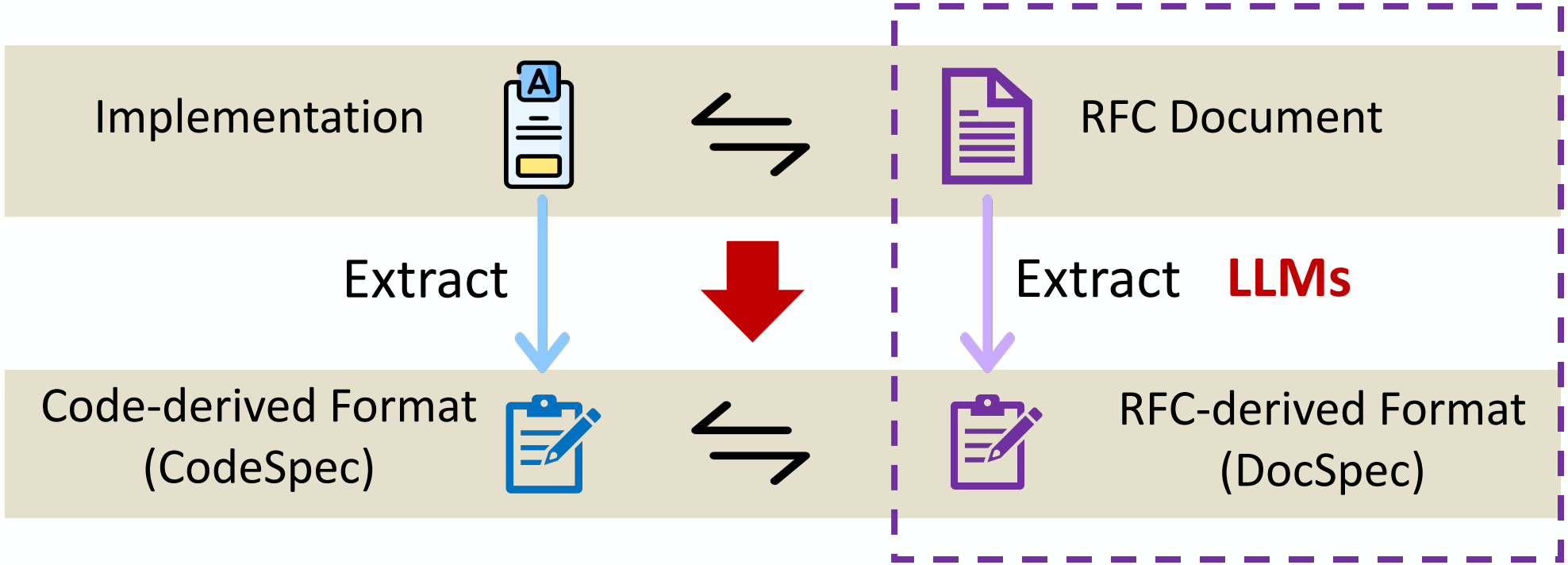


Use LLMs to bridge the gap between unstructured natural language and source code

PURDUE UNIVERSITY

Our Solution: *Doc. vs. Impl. => Format Spec Comparison*

- Compare **formats** derived from the ***parser code*** and the ***official RFC***



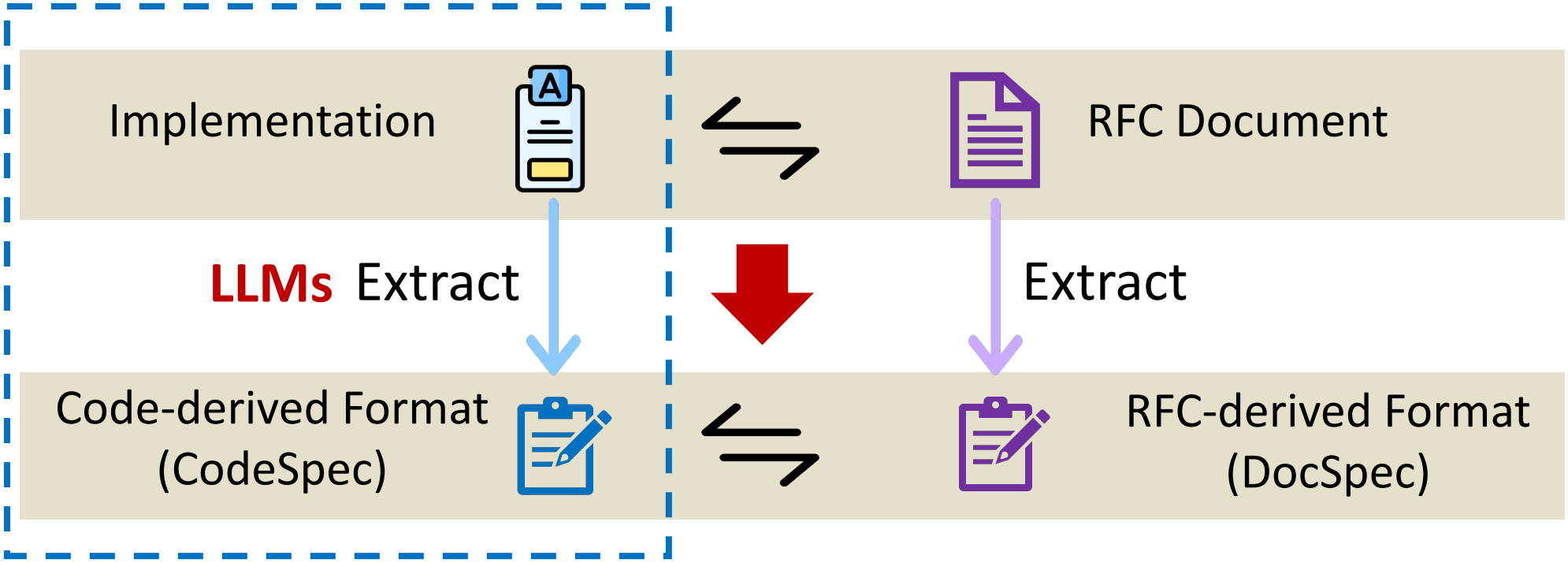
Already addressed by some recent paper: 3DGen[1], ParCleanse[2]

PURDUE UNIVERSITY

[1] 3DGen: AI-Assisted Generation of Provably Correct Binary Format Parsers (ICSE '25)
[2] Validating Network Protocol Parsers with Traceable RFC Document Interpretation (ISSTA '25)

Our Solution: *Doc. vs. Impl. => Format Spec Comparison*

- Compare **formats** derived from the ***parser code*** and the ***official RFC***



PURDUE UNIVERSITY

Challenge 1: *How to automatically retrieve parsing context?*

LLMs: context window limits

Hallucinate a lot when input is too much!

…

Parsing Logic Is Not Isolated!

```
void bfd_recv_cb (struct thread *t) {
  ... /* Socket handling
  if (sd == bvrf->bg_shop || sd == bvrf->bg_mhop) {
      mlen = bfd_recv_ipv4(sd, msgbuf,
sizeof(msgbuf), ...);
  }
  ...

  if (mlen < BFD_PKT_LEN) {
      cp_debug("too small (%ld bytes)", mlen);
      return;
  }
  /* Parse the control header for inconsistencies:
  struct bfd_pkt *cp = (struct bfd_pkt *)(msgbuf);
  if (BFD_GETVER(cp->diag) != BFD_VERSION) {
      cp_debug("bad version %d", BFD_GETVER(cp->diag));
      return;
  }
  ...
```

Parsing Logic Is Not Isolated!

🚫 **Irrelevant Logic**

Socket checks: *runtime-specific, unrelated to buffer parsing*

Input reading: *specify which variable represents buffer and buffer length*

```
void bfd_recv_cb (struct thread *t) {
  ... /* Socket handling
  if (sd == bvrf->bg_shop || sd == bvrf->bg_mhop) {
      mlen = bfd_recv_ipv4(sd, msgbuf,
sizeof(msgbuf), ...);
  }
  ...

  if (mlen < BFD_PKT_LEN) {
      cp_debug("too small (%ld bytes)", mlen);
      return;
  }
  /* Parse the control header for inconsistencies:
  struct bfd_pkt *cp = (struct bfd_pkt *)(msgbuf);
  if (BFD_GETVER(cp->diag) != BFD_VERSION) {
      cp_debug("bad version %d", BFD_GETVER(cp->diag));
      return;
  }
  ...
```

Parsing Logic Is Not Isolated!

🚫 **Irrelevant Logic**

Socket checks: *runtime-specific, unrelated to buffer parsing*

Input reading: *specify which variable represents buffer and buffer length*

✅ **Parsing-Relevant Logic**

```
void bfd_recv_cb (struct thread *t) {
  ... /* Socket handling
  if (sd == bvrf->bg_shop || sd == bvrf->bg_mhop) {
      mlen = bfd_recv_ipv4(sd, msgbuf,
sizeof(msgbuf), ...);
  }
  ...

  if (mlen < BFD_PKT_LEN) {
      cp_debug("too small (%ld bytes)", mlen);
      return;
  }
  /* Parse the control header for inconsistencies:
  struct bfd_pkt *cp = (struct bfd_pkt *)(msgbuf);
  if (BFD_GETVER(cp->diag) != BFD_VERSION) {
      cp_debug("bad version %d", BFD_GETVER(cp->diag));
      return;
  }
  ...
```

PURDUE UNIVERSITY

Challenges

- CH1: *How to automatically retrieve parsing context?*
  - ➢ Analyze data/control dependency, function calls…
  - ➢ Identify parsing relevant code and irrelevant code

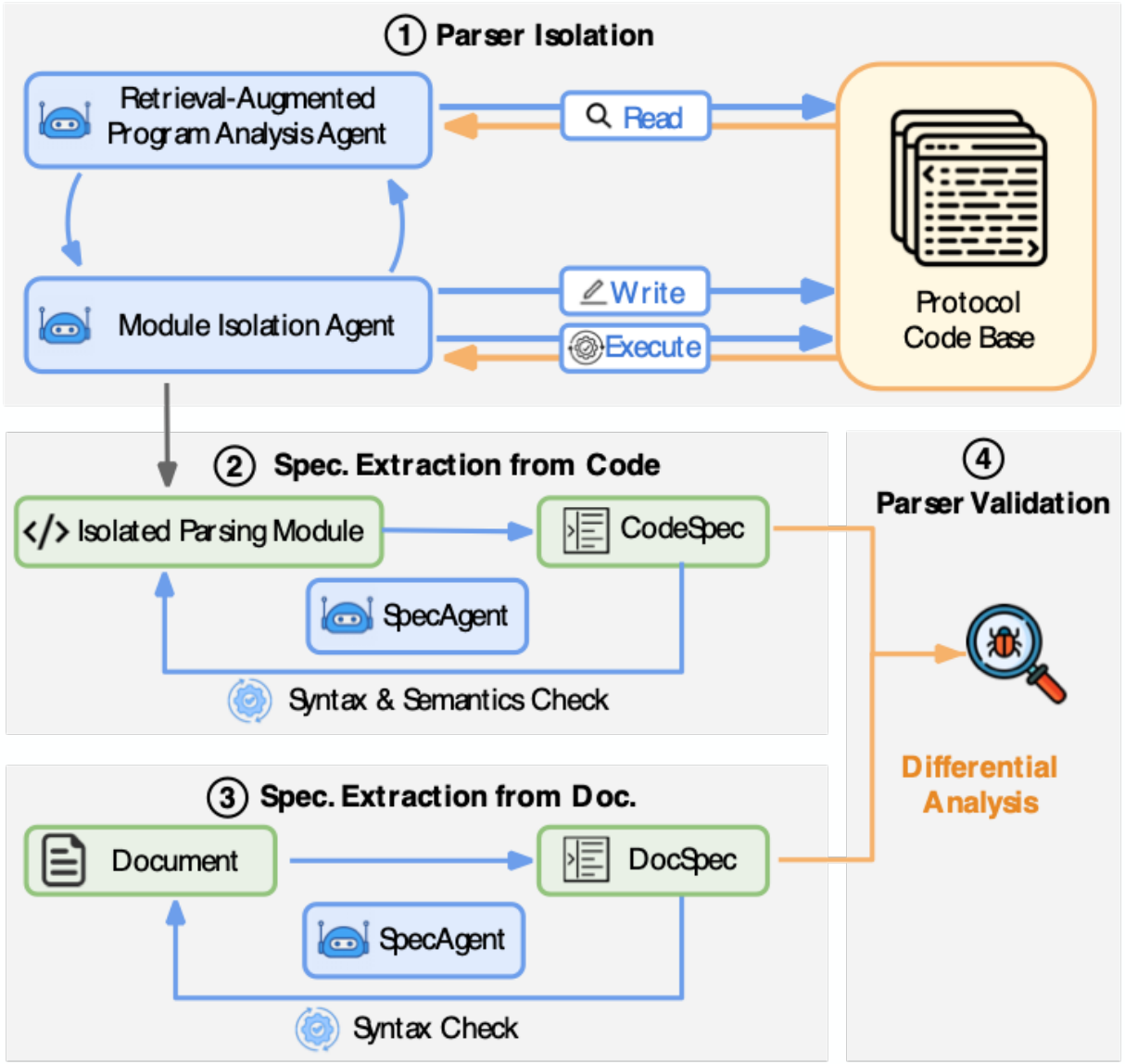
Challenges

- CH1: *How to automatically retrieve parsing context?*
  - ➢ Analyze data/control dependency, function calls…
  - ➢ Identify parsing relevant code and irrelevant code

- CH2: *How to extract accurate formats from code?*
  - ➢ Mitigate LLM hallucinations
  - ➢ construct a feedback loop for format refinement

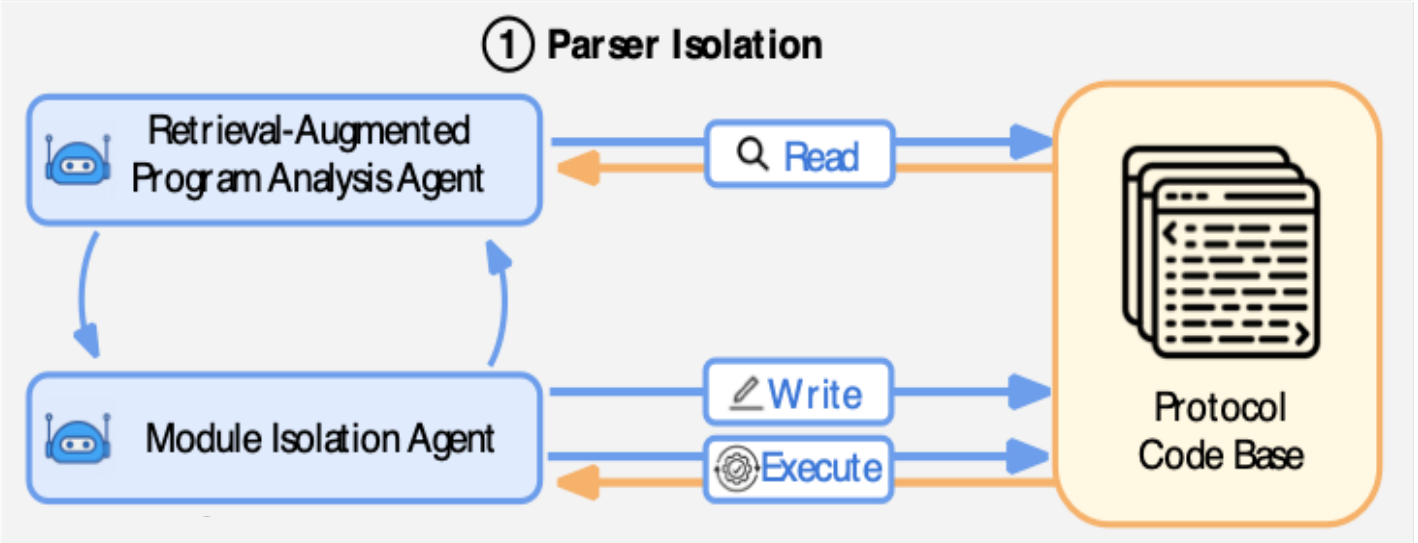
ParVAL: An LLM-based Parser Validation Framework

- **Goal**: Automatically validate parser implementations against RFCs



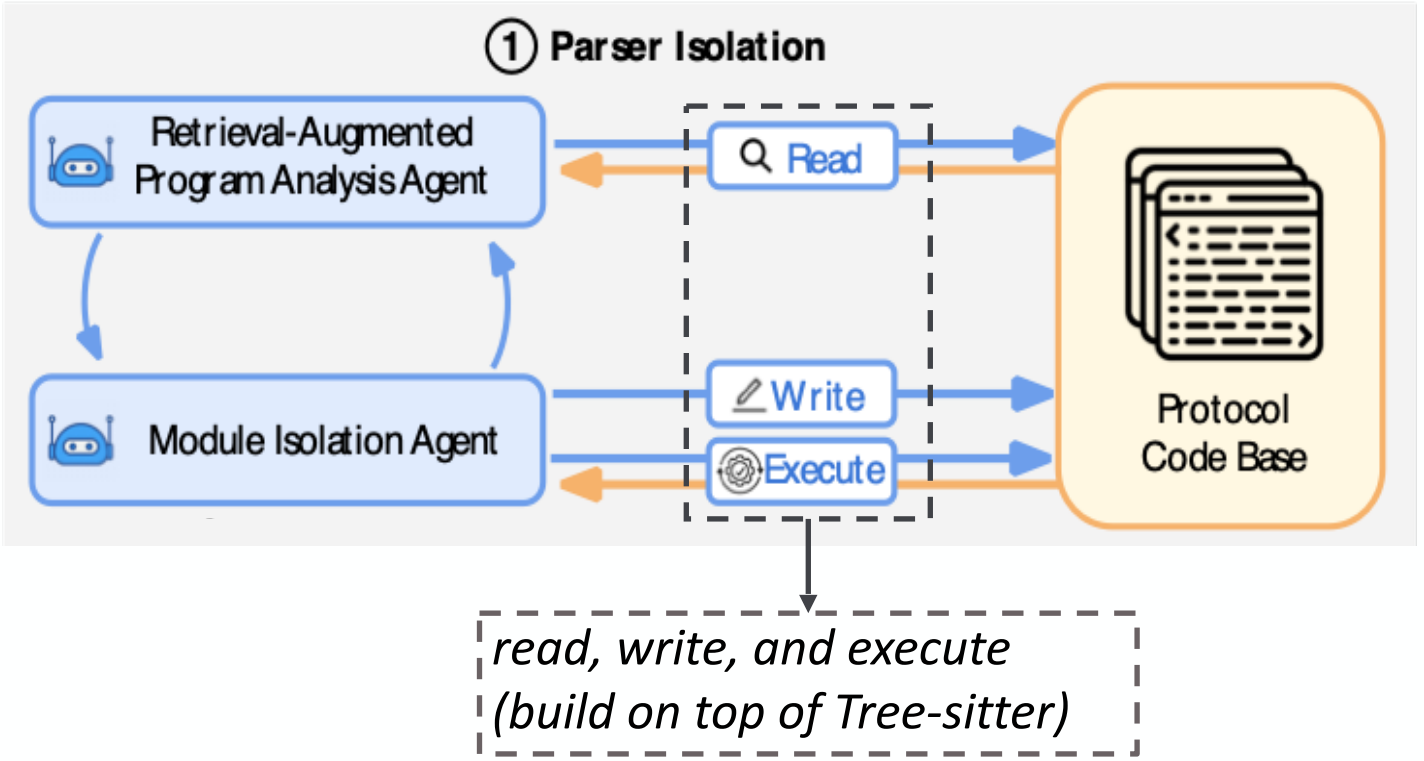
Step 1: Parser Isolation

- Two LLM Agents + Code Base Interaction Tools



Step 1: Parser Isolation

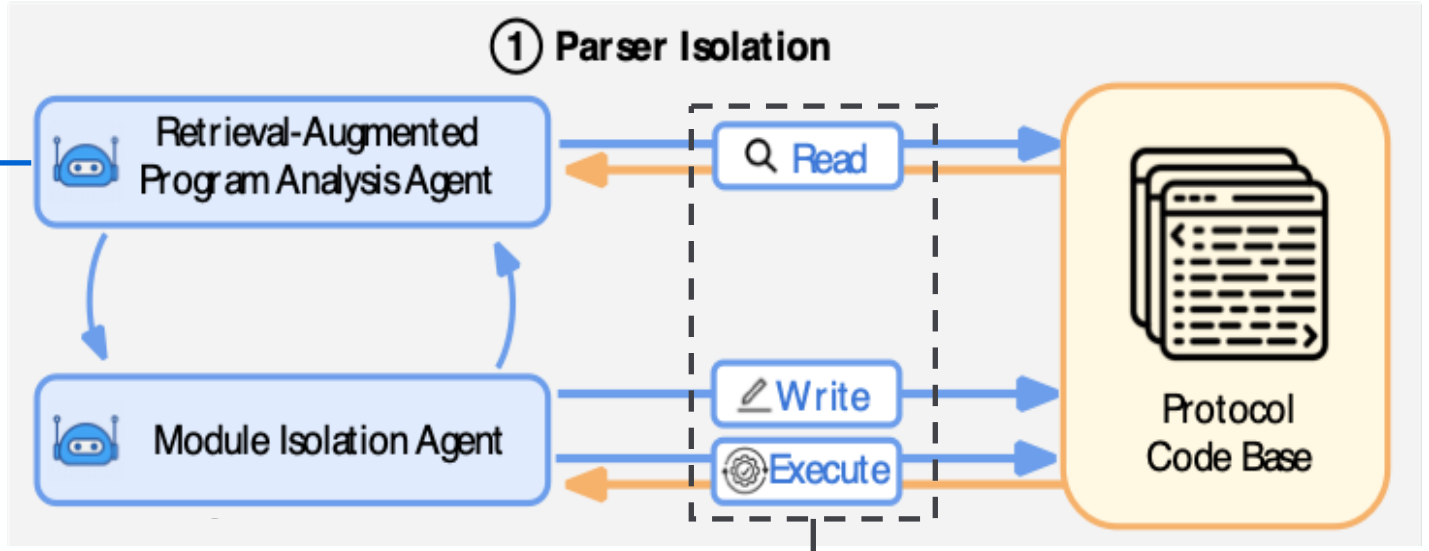
- Two LLM Agents + Code Base Interaction Tools



Step 1: Parser Isolation

- Two LLM Agents + Code Base Interaction Tools

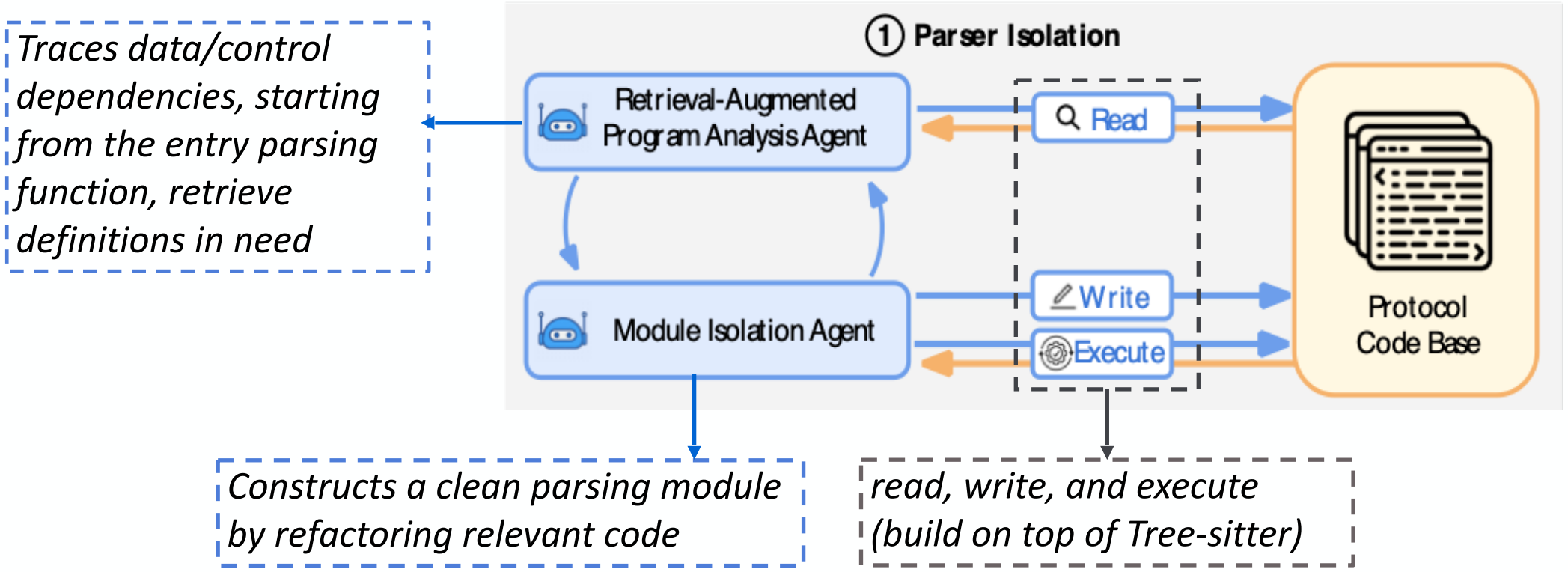
Traces data/control dependencies, starting from the entry parsing function, retrieve definitions in need



read, write, and execute (build on top of Tree-sitter)

Step 1: Parser Isolation

- Two LLM Agents + Code Base Interaction Tools



Case Study: BFD Implementation in FRRouting

Retrieval-Augmented Program Analysis Agent

🚫 **Irrelevant Logic (Exclude)**

Interaction Tool:
Read *definitions of* `bfd_recv_ipv4,…`

✅ **Parsing-Relevant Logic (Keep)**

```
void bfd_recv_cb (struct thread *t) {
  ... /* Socket handling
  if (sd == bvrf->bg_shop || sd == bvrf->bg_mhop) {
      mlen = bfd_recv_ipv4(sd, msgbuf,
sizeof(msgbuf), ...);
  }
  ...

  if (mlen < BFD_PKT_LEN) {
      cp_debug("too small (%ld bytes)", mlen);
      return;
  }
  /* Parse the control header for inconsistencies:
  struct bfd_pkt *cp = (struct bfd_pkt *)(msgbuf);
  if (BFD_GETVER(cp->diag) != BFD_VERSION) {
      cp_debug("bad version %d", BFD_GETVER(cp->diag));
      return;
  }
  ...
```

PURDUE UNIVERSITY

✅ Output: An Isolated Parsing Module

- Takes only `msgbuf` and `mlen`
- Returns a boolean indicating parsing success
- Excludes unrelated logic (e.g., socket handling)

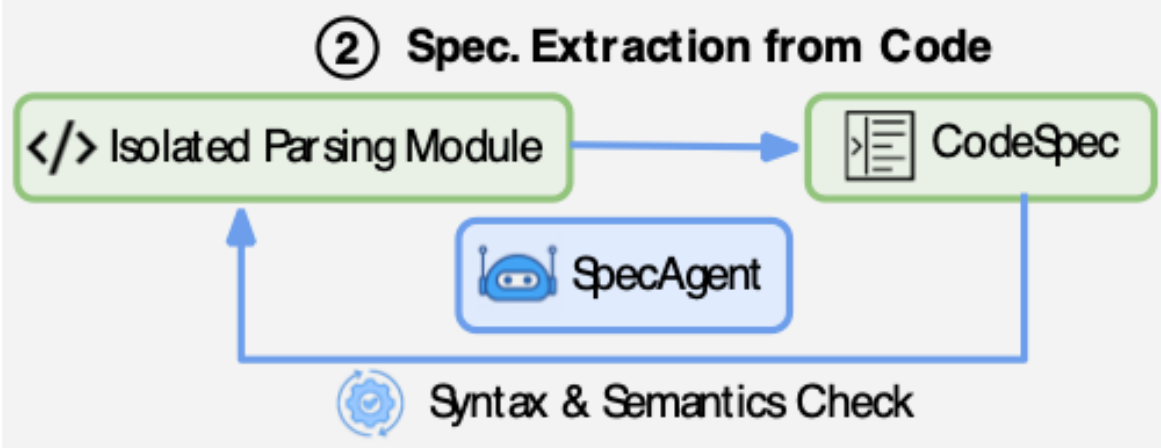
```
 bool parse_bfd_packet (uint8_t *msgbuf, size_t
mlen) {
   if (mlen < BFD_PKT_LEN) {
       cp_debug("too small (%ld bytes)", mlen);
              return false;
   }

   /* Parse the control header for inconsistencies:
   struct bfd_pkt *cp = (struct bfd_pkt *)(msgbuf);
   if (BFD_GETVER(cp->diag) != BFD_VERSION) {
       cp_debug("bad version %d", BFD_GETVER(cp-
>diag));              return false;
   }
   ... // other parsing logic
   return true;
 }
```

PURDUE UNIVERSITY

Step 2: Extract Format Specification from Code

- **Goal**: Isolated parser => CodeSpec

- **SpecAgent**, an LLM that:
  - ➢ Extract macro/type/.. Definition
  - ➢Understand a DSL (e.g., 3D) description
  - ➢ Outputs a structured format using the DSL
  - ➢ Syntax check & semantic check



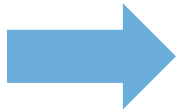
Step 2: Extract Format Specification from Code

- Semantic Check: generate symbolic test cases from **CodeSpec**
  - ➢Execute the isolated parsing module with test cases
  - ✅ **Positive test cases** (should pass)
  - ❌ **Negative test cases** (should fail)
- Compare behavior of the extracted spec with the actual module
  - ➢If mismatch → **SpecAgent** refine **CodeSpec** iteratively
  - ➢Feedback loop for CH2!

Case Study: CodeSpec

```
bool parse_bfd_packet (uint8_t *msgbuf, size_t
mlen) {
   if (mlen < BFD_PKT_LEN) {
       cp_debug("too small (%ld bytes)", mlen);
              return false;
   }

   /* Parse the control header for inconsistencies:
   struct bfd_pkt *cp = (struct bfd_pkt *)(msgbuf);
   if (BFD_GETVER(cp->diag) != BFD_VERSION) {
       cp_debug("bad version %d", BFD_GETVER(cp-
>diag));              return false;
   }
   ... // other parsing logic
   return true;
}
```

Isolated parsing module



```
entrypoint typedef struct _BFD () {
    UINT8BE Vers : 3 { Vers == 1 };
    UINT8BE Diag : 5;
    …
    UINT32BE
    RequiredMi                    rval;
} BFD;
```

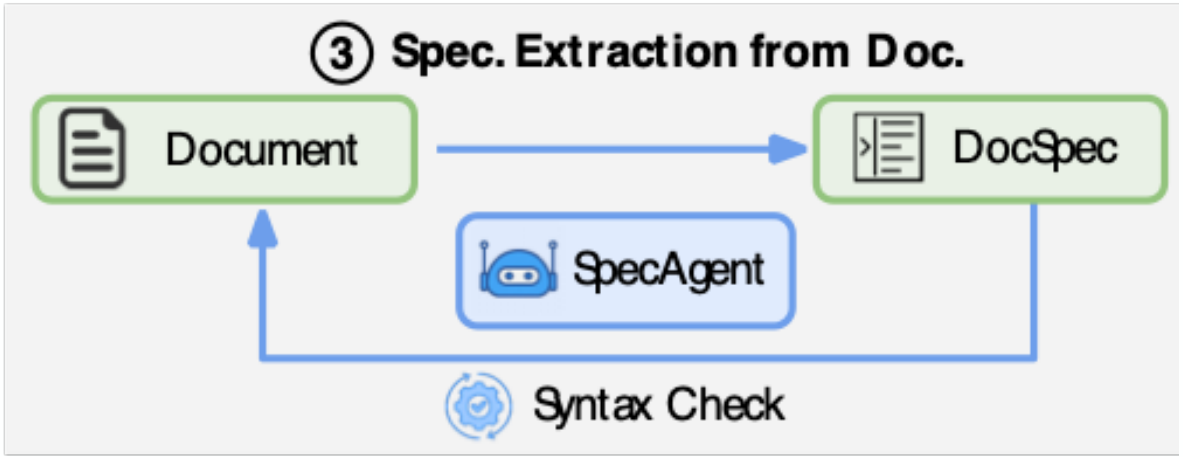
CodeSpec

PURDUE UNIVERSITY

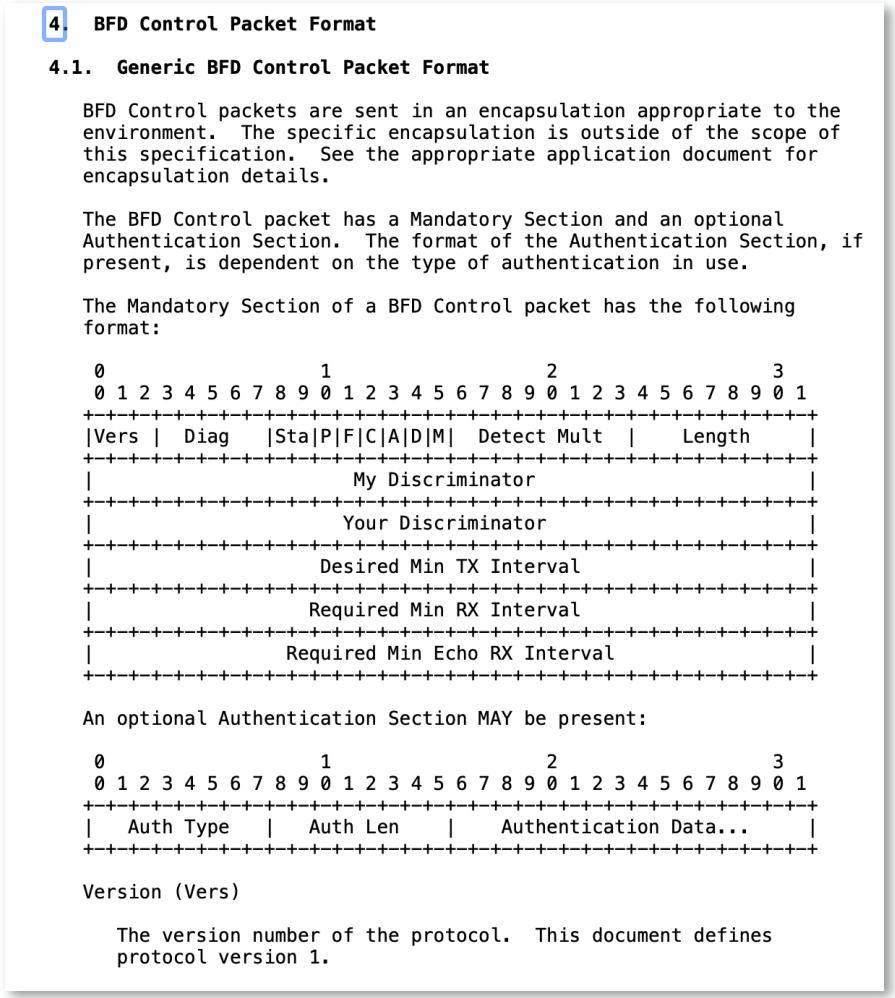
Step 3: Extract Format Specification from RFC

- **Goal**: RFC => DocSpec

- **SpecAgent**, an LLM that:
  - ➢ Understand natural language RFC
  - ➢ Understand a DSL (same with stage 2) description
  - ➢ Outputs a structured format using the DSL
    -- Enable uniform representation for comparison
  - ➢ Syntax check + semantic check (self-reflection)



DocSpec from RFC 5880



4. BFD Control Packet Format

4.1. Generic BFD Control Packet Format

BFD Control packets are sent in an encapsulation appropriate to the environment. The specific encapsulation is outside of the scope of this specification. See the appropriate application document for encapsulation details.

The BFD Control packet has a Mandatory Section and an optional Authentication Section. The format of the Authentication Section, if present, is dependent on the type of authentication in use.

The Mandatory Section of a BFD Control packet has the following format:

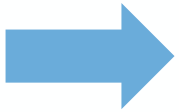
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Vers |  Diag   |Sta|P|F|C|A|D|M|  Detect Mult  |    Length     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       My Discriminator                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Your Discriminator                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Desired Min TX Interval                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Required Min RX Interval                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Required Min Echo RX Interval                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

An optional Authentication Section MAY be present:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Auth Type   |   Auth Len    |    Authentication Data...     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version (Vers)

The version number of the protocol. This document defines protocol version 1.



```
casetype _AuthSection (UINT8BE AuthType)
{
  switch(AuthType) {
    case 1: ..
} AuthPayload;

casetype _Auth(UINT8BE Auth) {
  switch(Auth){
    case 0: unit Empty;
    case 1: struct {
      UINT8BE AuthType;
      AuthPayload(AuthType) payload;
    } authsection;
  }
} Auth;

entrypoint typedef struct _BFD() {
      UINT8BE Vers : 3 { Vers == 1 };
      UINT8BE Diag : 5;
      …
      UINT32BE
    RequiredMinEchoRXInterval;
      Auth(A) a…;
} BFD;
```

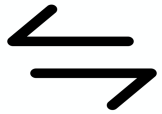
PURDUE UNIVERSITY

RFC 5880 **DocSpec**

Step 4: Diff CodeSpec & DocSpec

```
entrypoint typedef struct _BFD () {
    UINT8BE Vers : 3 { Vers == 1 };
    UINT8BE Diag : 5;
    …
    UINT32BE
    RequiredM                erval;
} BFD;
```

CodeSpec



```
casetype _AuthSection (UINT8BE AuthType)
{
  switch(AuthType) {
    case 1: ..
} AuthPayload;

casetype _Auth(UINT8BE Auth) {
  switch(Auth){
    case 0: unit Empty;
    case 1: struct {
      UINT8BE AuthType;
      AuthPayload(AuthType) payload;
    } authsection;
  }
} Auth;

entrypoint typedef struct _BFD() {
      UINT8BE Vers : 3 { Vers == 1 };
      UINT8BE Diag : 5;
      …
      UINT32BE
    RequiredMin                erval;
      Auth(A) a                ;
```

DocSpec

No authentication in the implementation!

PURDUE UNIVERSITY

Implementation & Evaluation

- ParVAL is built with:
  - ➢ AutoGen, a multi-agent framework for developing LLM applications
  - ➢ LLM API: Claude- 3.5 Sonnet, temperature: 0
  - ➢ Repository interaction: Tree-sitter, an AST-based parser to analyze and manipulate source code
  - ➢ Format DSL: 3D language, syntax checker: EverParse

- PARVAL uncovered **7 unique bugs**, **5 of which were previously unknown**

No.	Bug Description	New
1	Flag `M` should always be 0.	✗
2	Miss check Authentication Present (A) to handle optional Authentication Section.	✗
3	Miss validation for Simple Password Authentication Section format.	✓
4	Miss validation for Keyed MD5 Authentication Section format.	✓
5	Miss validation for Meticulous Keyed MD5 Authentication Section format.	✓
6	Miss validation for Keyed SHA1 Authentication Section Format.	✓
7	Miss validation for Meticulous Keyed SHA1 Authentication Section format.	✓

More technical details are available in our paper

Large Language Models for Validating Network Protocol Parsers

Mingwei Zheng
Department of Computer Science
Purdue University
West Lafayette, USA
zheng618@purdue.edu

Danning Xie
Department of Computer Science
Purdue University
West Lafayette, USA
xie342@purdue.edu

Xiangyu Zhang
Department of Computer Science
Purdue University
West Lafayette, USA
xyzhang@purdue.edu

***Abstract*—Network protocol parsers are essential for enabling correct and secure communication between devices. Bugs in these parsers can introduce critical vulnerabilities, including memory corruption, information leakage, and denial-of-service attacks. An intuitive way to assess parser correctness is to compare the implementation with its official protocol standard. However, this comparison is challenging because protocol standards are typically written in natural language, whereas implementations are in source code. Existing methods like model checking, fuzzing, and differential testing have been used to find parsing bugs, but they either require significant manual effort or ignore the protocol standards, limiting their ability to detect semantic violations. To enable more automated validation of parser implementations against protocol standards, we propose PARVAL, a multi-agent framework built on large language models (LLMs). PARVAL leverages the capabilities of LLMs to understand both natural language and code. It transforms both protocol standards and their implementations into a unified intermediate representation, referred to as format specifications, and performs a differential comparison to uncover inconsistencies. We evaluate PARVAL on the Bidirectional Forwarding Detection (BFD) protocol. Our experiments demonstrate that PARVAL successfully identifies inconsistencies between the implementation and its RFC standard, achieving a low false positive rate of 5.6%. PARVAL uncovers seven unique bugs, including five previously unknown issues.**

***Index Terms*—Network protocol parsing, format specifications, large language models**

including memory corruption, information leakage, and denial-of-service attacks. For instance, the Heartbleed [4] vulnerability in OpenSSL [5]'s TLS parser allowed attackers to access sensitive user information due to a missing bounds check. Similarly, CVE-2021-41773 [6] in Apache's HTTP server exposed confidential files through a path traversal flaw in the request parser.

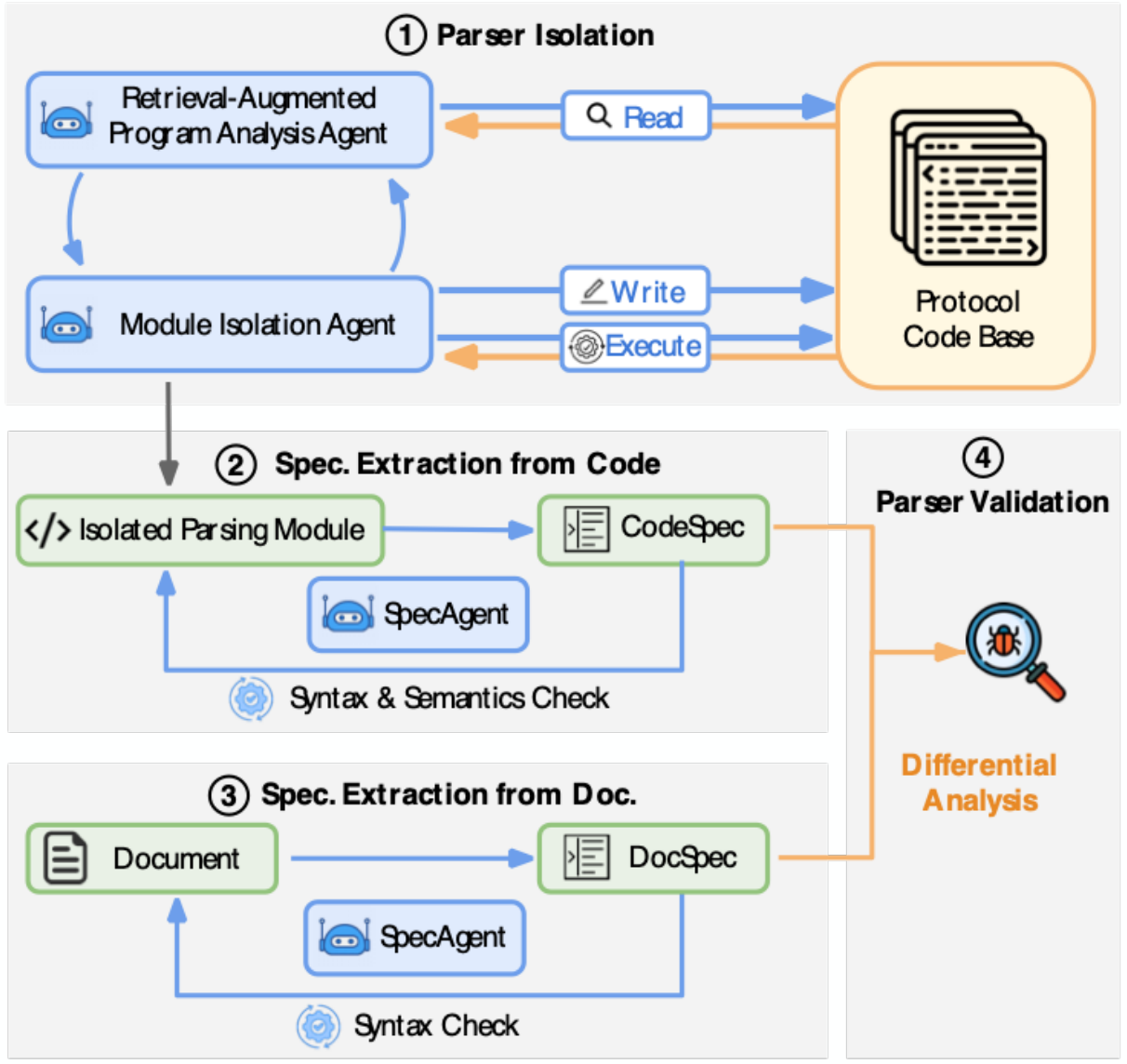
Validating the correctness of network protocol parsers is inherently challenging. A natural approach is to compare the parser implementation against its official protocol standard to identify inconsistencies. However, this process is complicated by a fundamental representational gap: protocol standards are typically written in natural language (e.g., RFCs), while implementations are written in source code. Although both aim to specify the same behavior, their different representations hinder direct comparison, making systematic validation a significant challenge.

Various techniques have been proposed to address this challenge. Model checking [7]–[9] verifies parser behavior against formal protocol models to detect logical errors. While effective, it often requires manual construction of formal models, making it impractical for applying to large and frequently evolving protocols. Fuzzing [10], [11] automatically produces large volumes of test inputs to expose memory-related issues, such as crashes. However, it primarily targets low-level bugs and fails to assess whether the parser conforms to the protocol's intended semantics, leaving many semantic bugs [12]–[15] undetected. Differential analysis [12], [16] identifies inconsistencies by comparing how different imple-

PURDUE UNIVERSITY

Large Language Models for Validating Network Protocol Parsers

Paper



Code

PURDUE UNIVERSITY