Microsoft

3DGen: *AI-assisted Generation of Provably Correct Binary Format Parsers*

Sarah Fakhoury, Markus Kuppe, Shuvendu Lahiri, Tahina Ramananandro, Nikhil Swamy
Microsoft Research

Secure Binary Data Parsing is Critical

- Parsing and input validation failures: A major root cause of software security vulnerabilities
- `80%, according to DARPA, MITRE
- Mostly due to handwritten parsing code
  - Especially disastrous in memory unsafe languages
  - Better in Rust, but still open to runtime panics and functional bugs
- Writing functionally correct parsers is hard
  - Endianness, data dependencies, size constraints, etc.

A First Step: Verified Parser Generation with everparse

- Abolish writing parser code by hand
- Instead, specify data formats in **3D**, a high-level declarative notation
- Auto-generate provably correct performant C code to parse binary messages
- Integrate with generated code with existing codebases

[1] Swamy, Nikhil, et al. "Hardening attack surfaces with formally proven binary format parsers." *International Conference on Programming Language Design and Implementation (PLDI)*. 2022.
[2] Ramananandro, Tahina, et al. "EverParse: Verified secure zero-copy parsers for authenticated message formats." *28th USENIX Security Symposium (USENIX Security 19)*. 2019

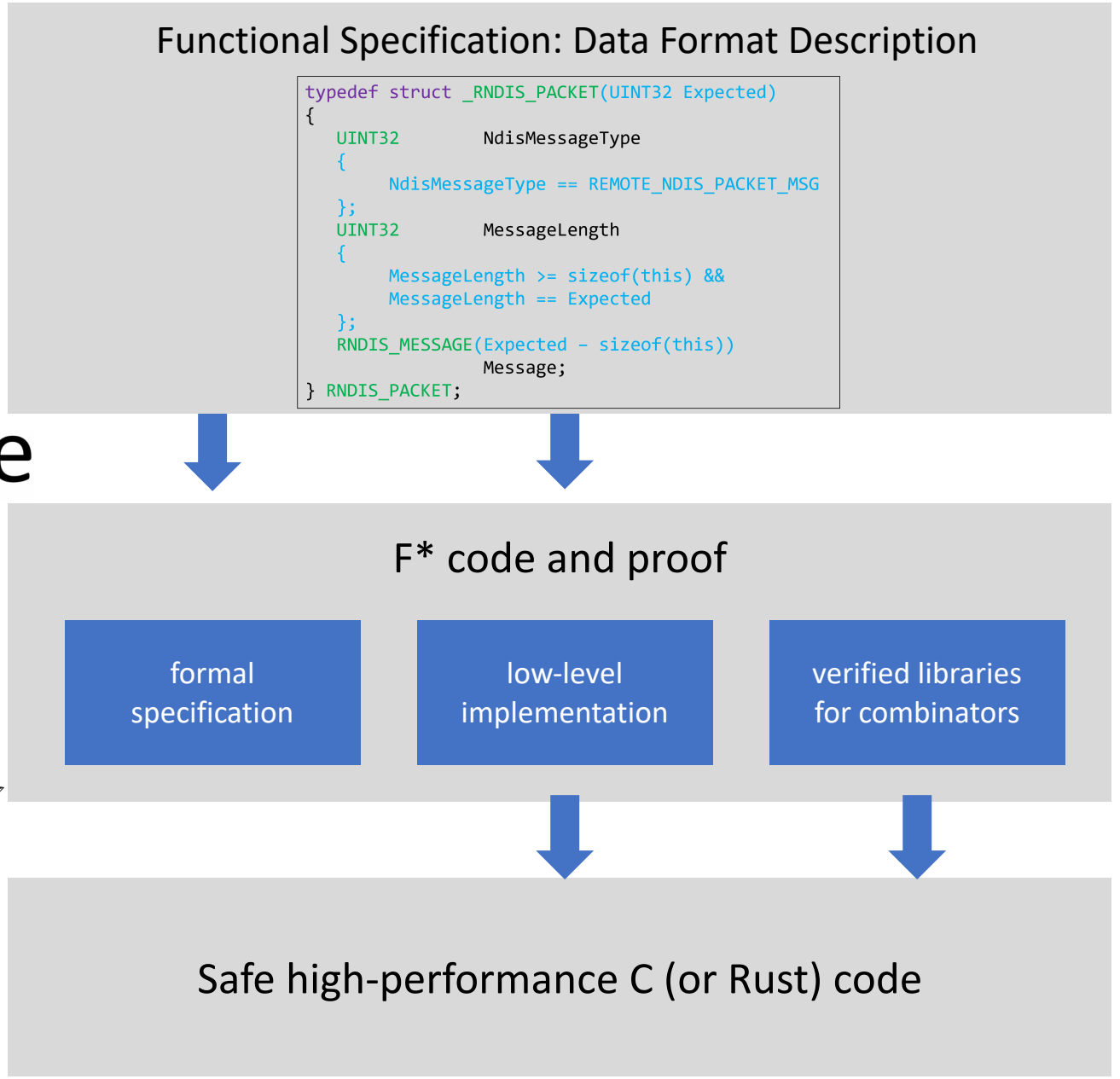
Starting from a language of message formats resembling C type definitions

EverParse auto-generates parsing code in C that is:

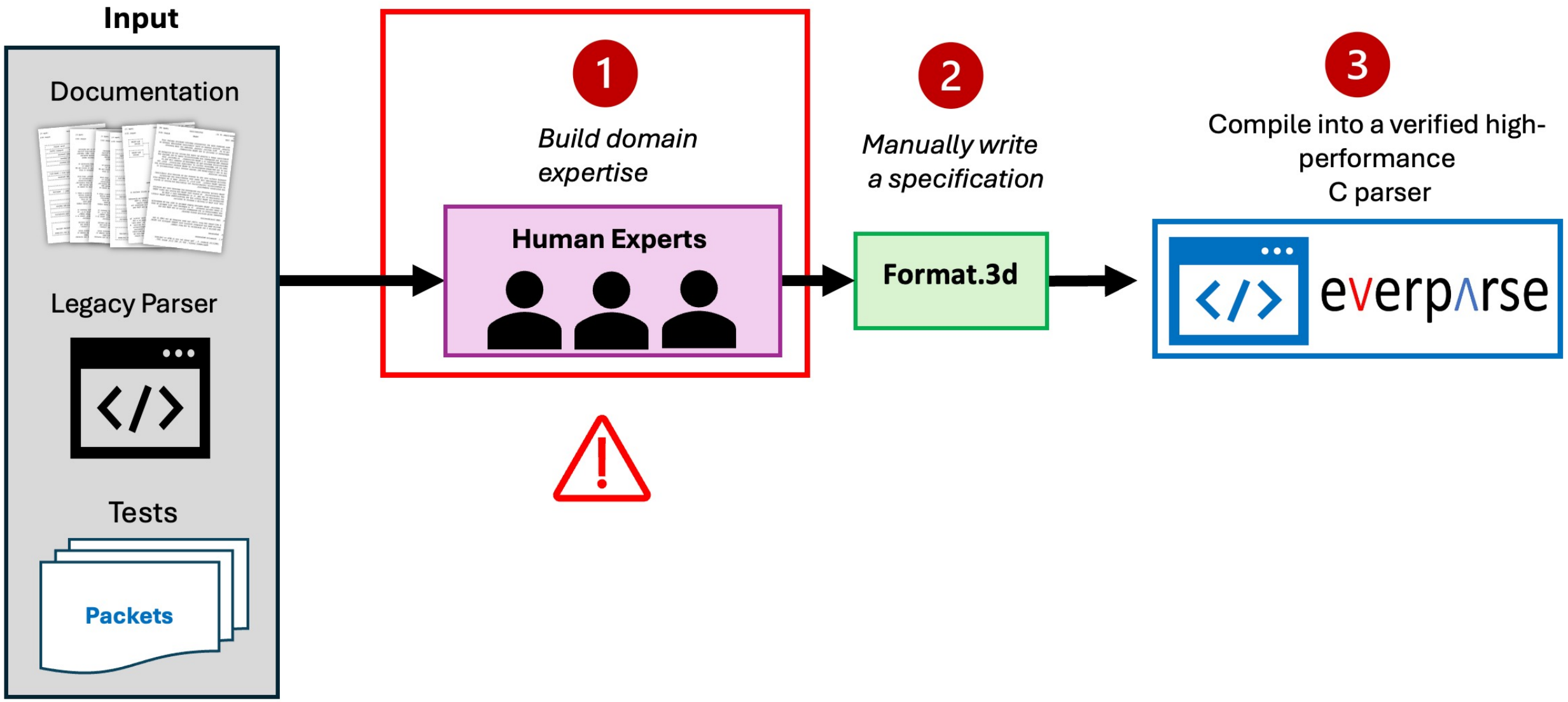
- Safe
- Functionally Correct
- Fast (zero-copy)
- Double-fetch free
- Portable

Correctness:
Accept only all well-formed messages

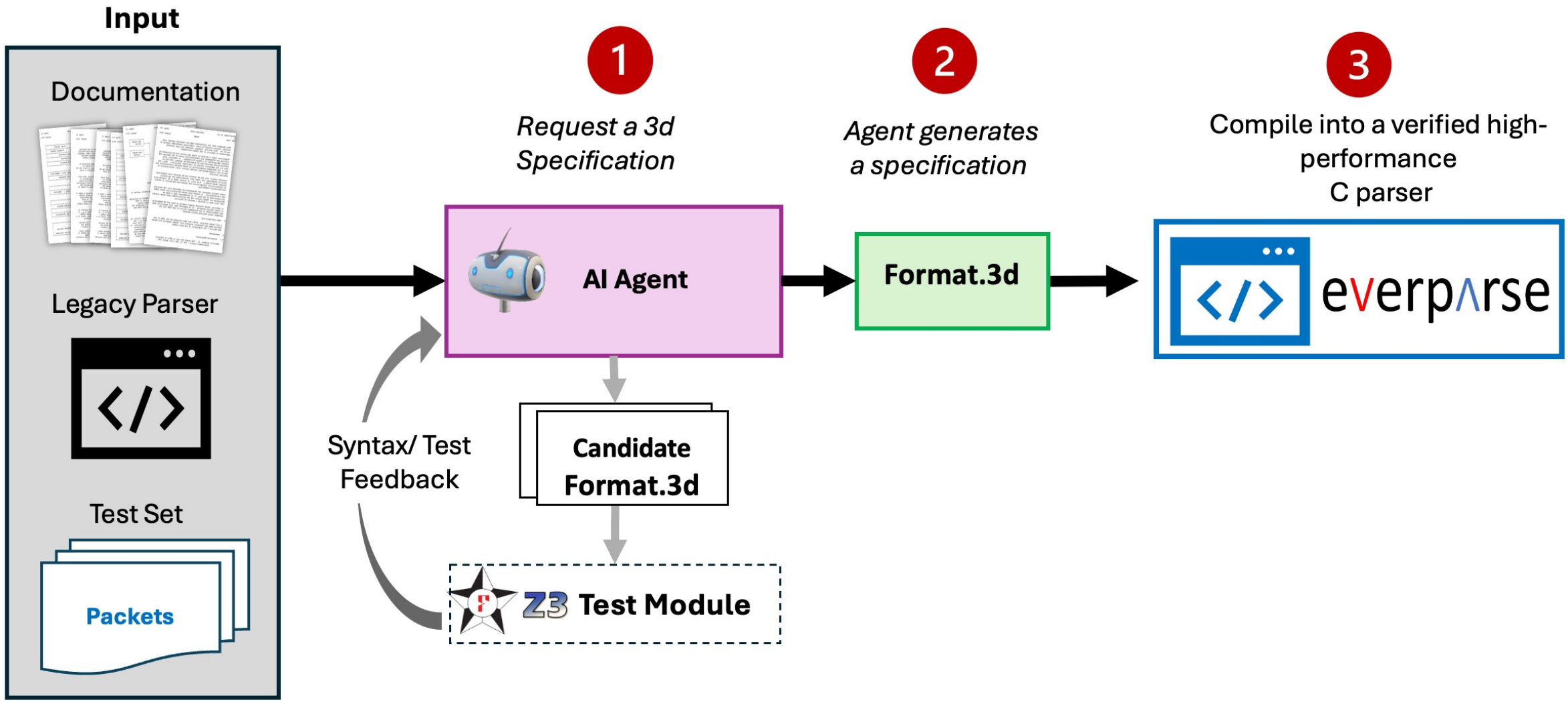
everparse



Current Approach

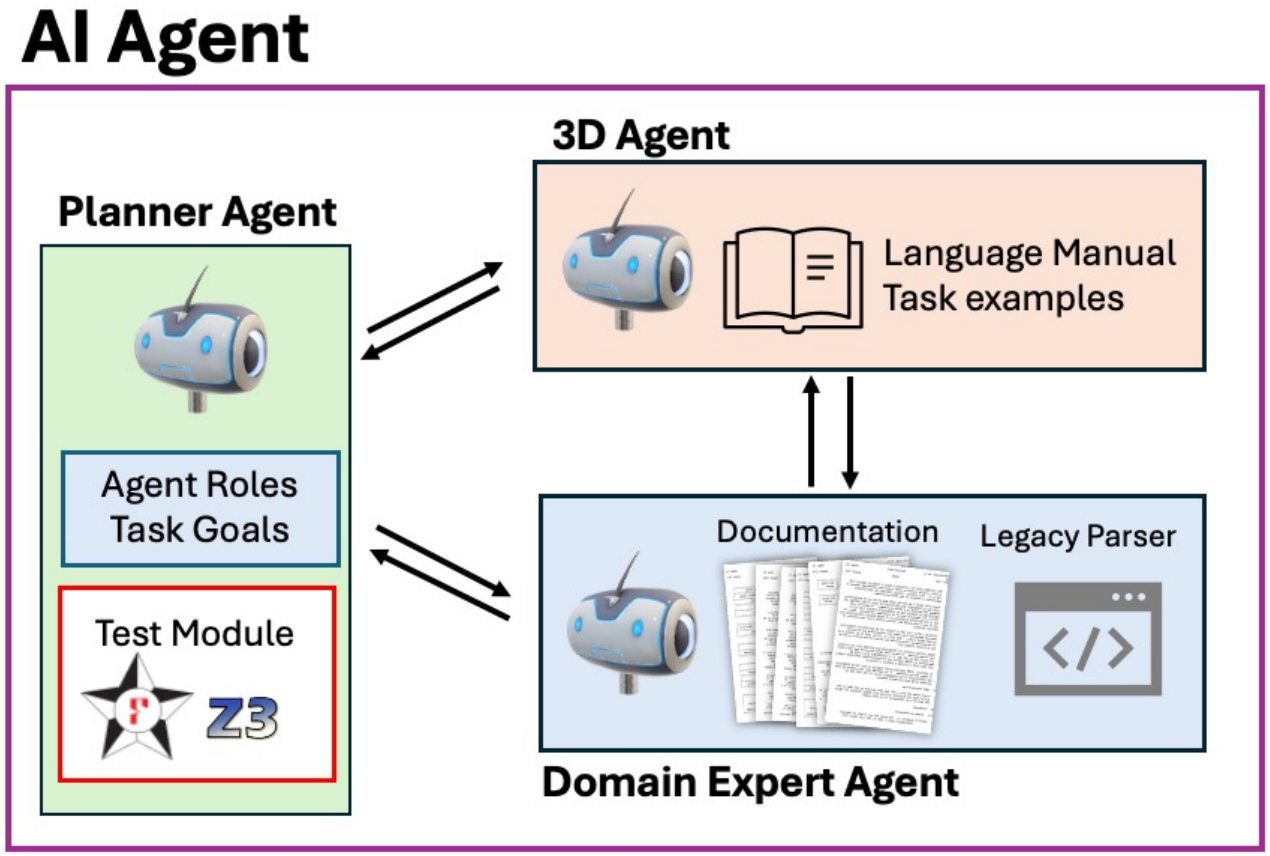


3DGen: AI-Assisted Generation of Verified Parsers



Agent Implementation

- Three Agent personas collaborating:
  - **Planner:** dictates roles, orchestrates conversation
  - **Domain Expert Agent**: Extracts constraints from NL or Code, provides feedback about generated specification
  - **3D Agent:** translates extracted specifications into 3D
- Implemented with AutoGen [1]
  - Composable Retrieval Augmented (RAG) agents
- No fine-tuning, easy migration to GPT-X!
  - Gpt-4-32k model



[1] Wu, Qingyun, et al. "AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework." *arXiv preprint arXiv:2308.08155* (2023)

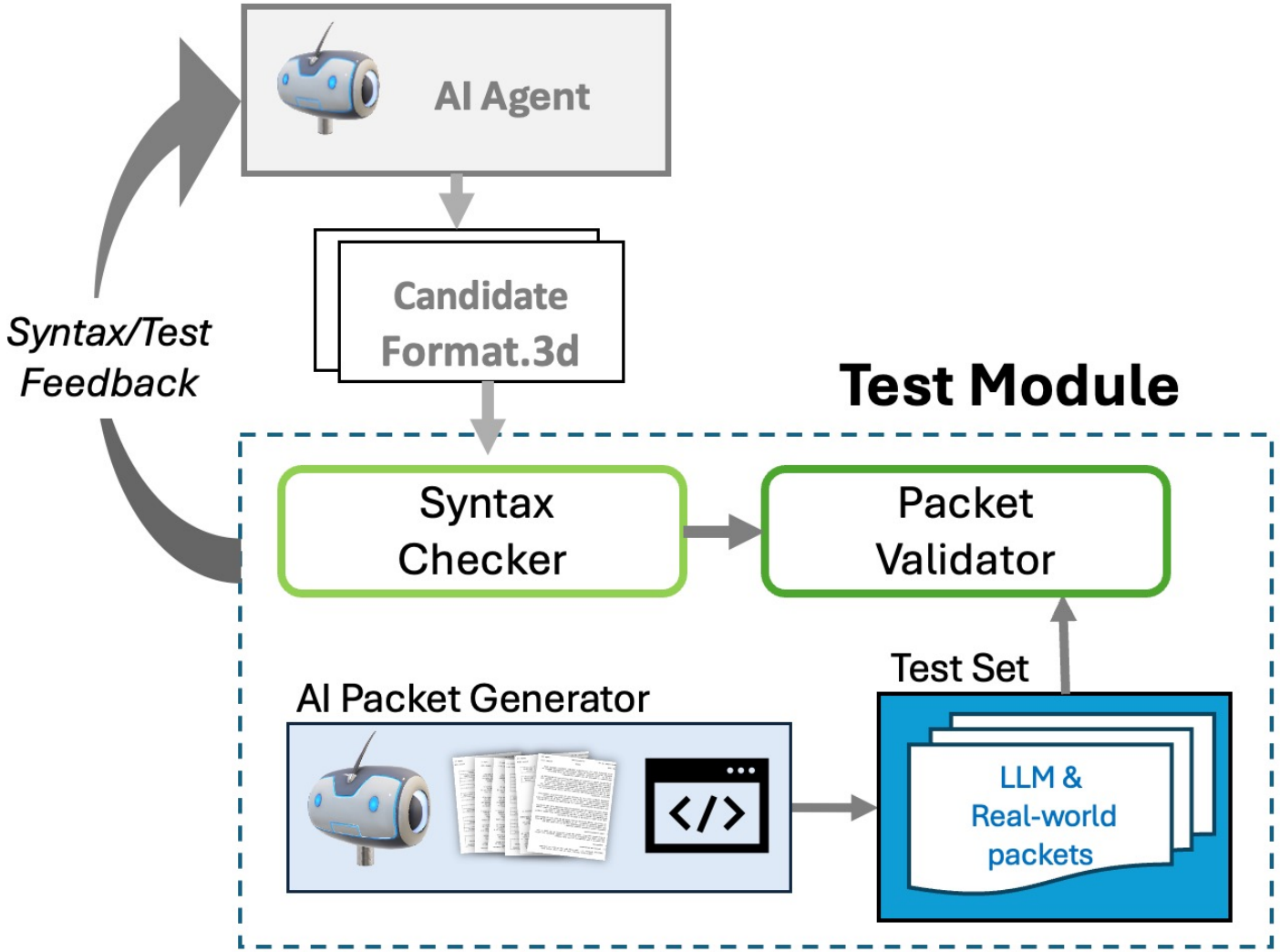
Test Module

Syntax Checker

- Feedback from 3D language parser

Packet Validator

- Test candidate specifications against a large set of (+ve/-ve) packets

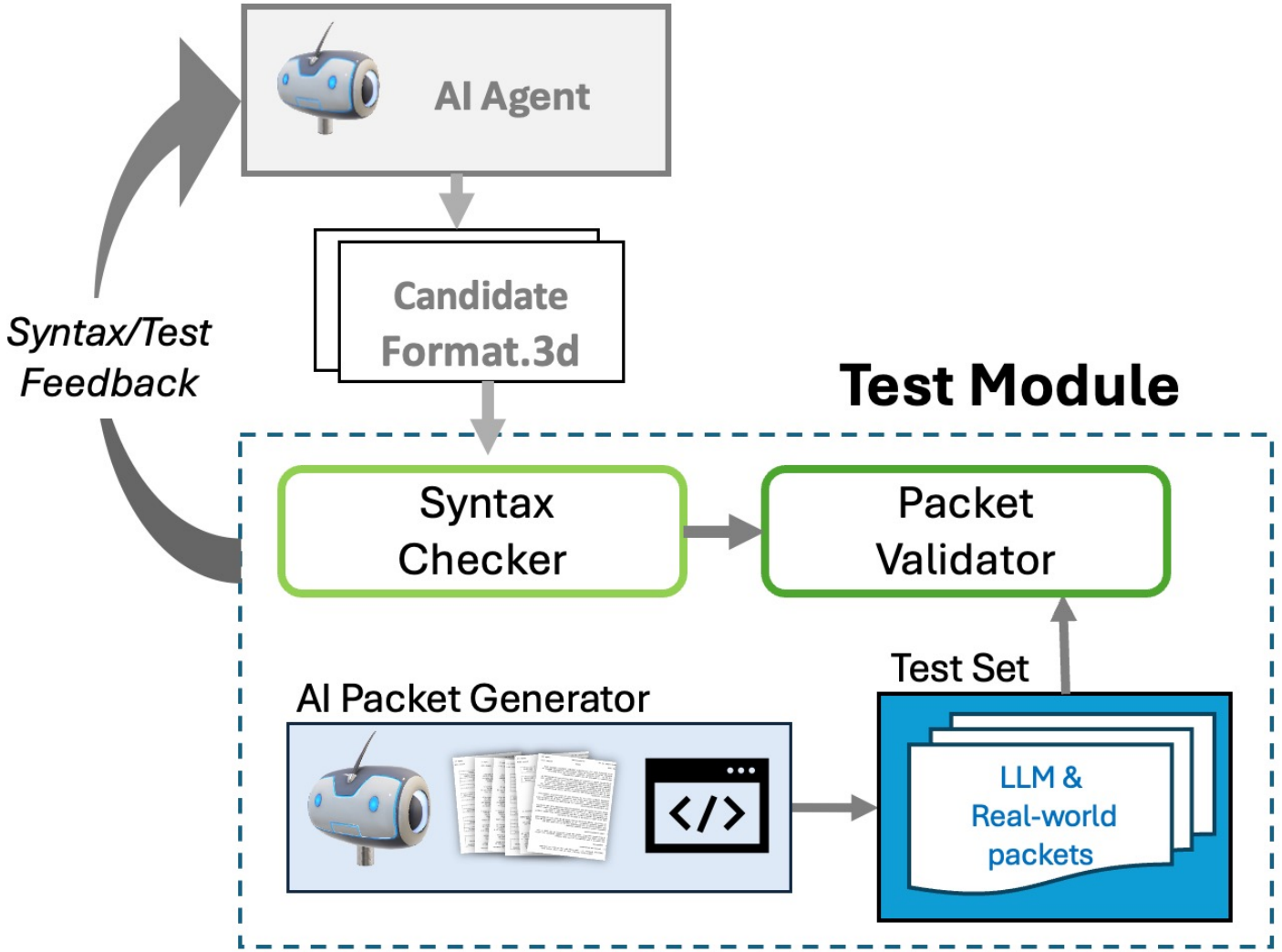


Test Module

Test Set Construction:

- **Existing real-world packets**
- **Z3 packet generator**
  - Input: candidate 3D specification compiled into SMT2
  - Output: sample packets
- **LLM packet generator**
  - Input: RFC
  - Output: Sample packets

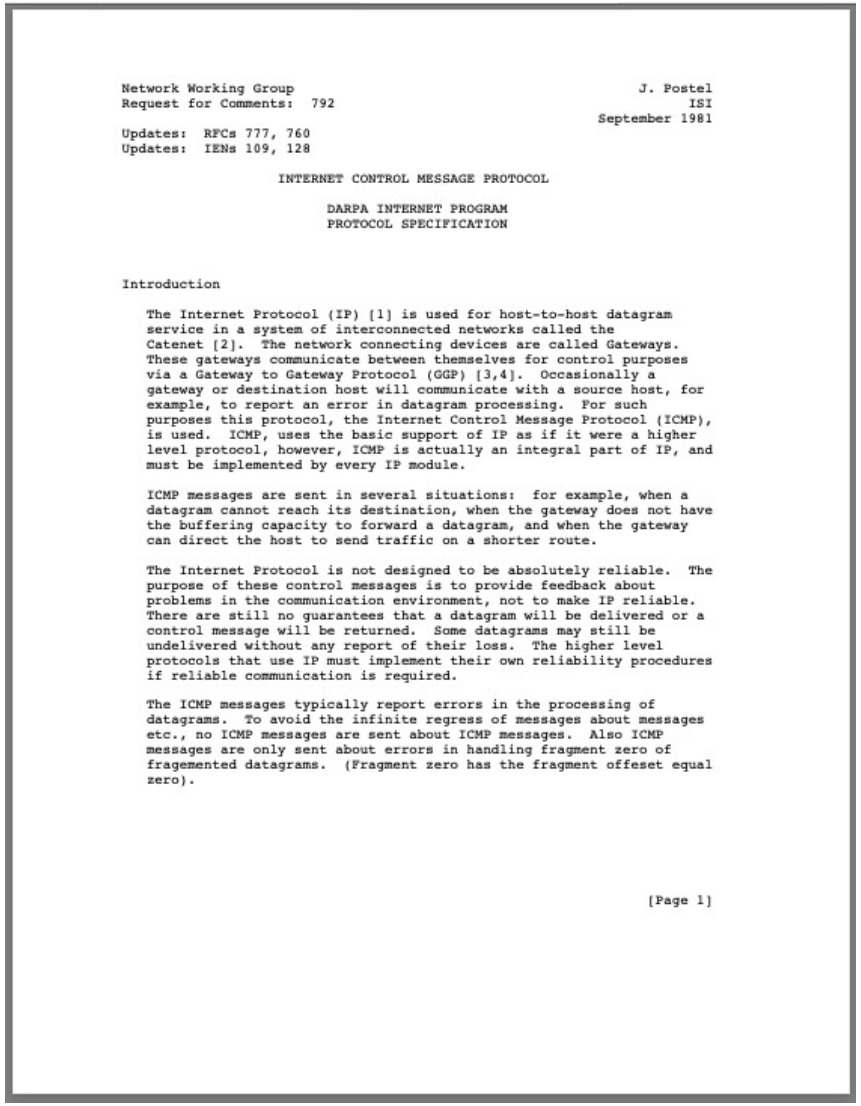
All packets labeled with legacy parser



Example Walkthrough: Internet Control Message Protocol (ICMP)

- **21 Pages of specifications**

- **8 message types**
  - Destination Unreachable Message
  - Time Exceeded Message
  - Parameter Problem Message
  - Source Quench Message
  - Redirect Message
  - Echo or Echo Reply Message
  - Timestamp or Timestamp Reply Message
  - Information Request or Information Reply Message



Network Working Group
Request for Comments: 792

J. Postel
ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

Introduction

The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling fragment zero of fragemented datagrams. (Fragment zero has the fragment offeset equal zero).

[Page 1]

Microsoft

Planner Agent

Agent Roles
Task Goals

Test Module

1. **Understand the RFC:** First, we need to understand the RFC and its structure. The RFC describes the Internet Control Message Protocol (ICMP), which is used for error reporting and diagnostics in the IP layer. The ICMP messages are encapsulated within IP datagrams and have a specific format.

2. **Identify Relevant Parts:** Not all of the RFC is relevant to our task. We are interested in the "Message Formats" section, which describes the different types of ICMP messages and their structures. Specifically, we need to translate the specifications of these message formats into 3D code.

3. **Decompose the RFC:** We will break down the relevant parts of the RFC into smaller pieces. For each type of ICMP message, we will identify the fields and their properties (e.g., type, length, and purpose).

4. **Translate to 3D Code:** For each type of ICMP message, we will write a corresponding 3D struct. Each struct will contain fields that match the ones described in the RFC. The type and constraints of each field will be based on the properties identified in the previous step.

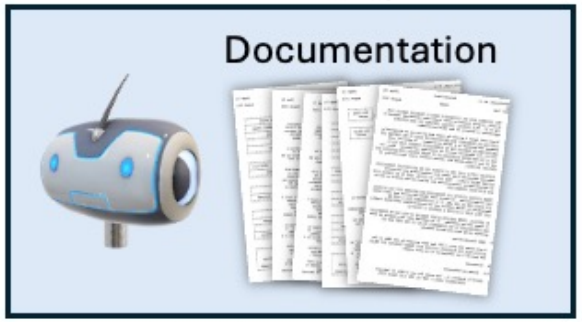
5. **Verify the Code:** We will use the `functions.test_module` tool to execute the 3D code and verify that it correctly implements the specifications in the RFC. If any errors or issues are found, we will revise the code accordingly.

7. **Repeat Steps 3-6:** We will repeat these steps for each type of ICMP message described in the RFC.

8. **RFC Expert Review:** Once all the message types have been translated to 3D code, the RFC expert will review the code to ensure that it accurately captures the specifications in the RFC.

9. **Finalize the Code:** After the RFC expert has approved the code, we will finalize it and complete the task.

Microsoft



Domain Expert Agent

To implement the plan described, let's start by understanding the key elements of the provided ICMP (Internet Control Message Protocol) RFC 792 document. The main messages specified are:

- ICMP Header (common to all ICMP messages)
- Destination Unreachable Message
- Time Exceeded Message
- Parameter Problem Message
- Source Quench Message
- Redirect Message
- Echo or Echo Reply Message
- Timestamp or Timestamp Reply Message
- Information Request or Information Reply Message.

Every message has some common elements, i.e., a 'Type', 'Code', and 'Checksum'. Moreover, every message after these fields has a specific structure depending on its type. Thus, **we should first define these classes for the general ICMP header and then create specific types using the `casetype` declaration in 3D for each type of ICMP message.**

Refinement: packet feedback

Test Module Feedback

```
>>>>>>>> EXECUTING FUNCTION test_specification...

***** Response from calling function "test_specification" *****
Code did not correctly parse protocol packets:
      Witness from ./3DGen/packets/binary_files/ICMP/Echo_Reply_0.dat REJECTED because
      validator only consumed 9 out of 40 bytes
      Parsing failed at position 5: type name _Echo, field name Data. Reason:
      constraint failed
**************************************************************
```

Specification fails on test set!

Refinement: packet feedback

```
Echo or Echo Reply Message

    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Type      |     Code      |            Checksum           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Identifier          |        Sequence Number        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Data ...
   +-+-+-+-+-+-
```

3D Agent

The error is due to the incorrect Data declaration `UINT8 Data;` The size of the `Data` field is not specified in the RFC. We can assume that the `Data` field takes up the rest of the packet. In 3d, we can specify this by using the `[:consume-all]` notation, which means that the array extends to the end of the packet. Let's correct this and try again.

Refinement: packet feedback


```
typedef struct _ECHO {
  UINT8 Code { Code == 0 };
  UINT16BE Checksum;
  UINT16BE Identifier;
  UINT16BE SequenceNumber;
-   UINT8 Data;
+   UINT8 Data[:consume-all];
} ECHO;
```

```
typedef struct _DEST_UNREACHABLE {…
typedef struct _PARAMETER_PROBLEM {…
} PARAMETER_PROBLEM;
typedef struct _SOURCE_QUENCH {…
} SOURCE_QUENCH;
typedef struct _REDIRECT {…
} REDIRECT;
typedef struct _ECHO {…
} ECHO;
typedef struct _TIMESTAMP {…
} TIMESTAMP;
typedef struct _INFO_REQUEST {…
} INFO_REQUEST;
```

```
typedef struct _TIME_EXCEEDED {
  UINT8 Code { Code == 0 || Code == 1 };
  UINT16BE Checksum;
  UINT32BE unused { unused == 0 };
  IPV4::IPV4_HEADER Header;
  UINT64BE DataDatagram;
} TIME_EXCEEDED;

casetype _ICMP_MESSAGE (UINT8 Type) {
  switch(Type) {
    case 3: DEST_UNREACHABLE DestUnreachable;
    case 11: TIME_EXCEEDED TimeExceeded;
    case 12: PARAMETER_PROBLEM ParameterProblem;
    case 4: SOURCE_QUENCH SourceQuench;
    case 5: REDIRECT RedirectMessage;
    case 8: ECHO Echo;
    case 0: ECHO EchoReply;
    case 13: TIMESTAMP Timestamp;
    case 14: TIMESTAMP Timestamp;
    case 15: INFO_REQUEST InfoReply;
    case 16: INFO_REQUEST InfoRequest;
  }
} ICMP_MESSAGE;

entrypoint typedef struct _ICMP_PACKET {
  UINT8 Type;
  ICMP_MESSAGE(Type) icmp_message;
} ICMP_PACKET;
```

3D Agent

Specification passes on all tests!

```
typedef struct _DEST_UNREACHABLE {…
typedef struct _PARAMETER_PROBLEM {…
} PARAMETER_PROBLEM;
typedef struct _SOURCE_QUENCH {…
} SOURCE_QUENCH;
typedef struct _REDIRECT {…
} REDIRECT;
typedef struct _ECHO {…
} ECHO;
typedef struct _TIMESTAMP {…
} TIMESTAMP;
typedef struct _INFO_REQUEST {…
} INFO_REQUEST;
```

```
typedef struct _TIME_EXCEEDED {
  UINT8 Code { Code == 0 || Code == 1 };
  UINT16BE Checksum;
  UINT32BE unused { unused == 0 };
  IPV4::IPV4_HEADER Header;
  UINT64BE DataDatagram;
} TIME_EXCEEDED;

casetype _ICMP_MESSAGE (UINT8 Type) {
  switch(Type) {
    case 3: DEST_UNREACHABLE DestUnreachable;
    case 11: TIME_EXCEEDED TimeExceeded;
    case 12: PARAMETER_PROBLEM ParameterProblem;
    case 4: SOURCE_QUENCH SourceQuench;
    case 5: REDIRECT RedirectMessage;
    case 8: ECHO Echo;
    case 0: ECHO EchoReply;
    case 13: TIMESTAMP Timestamp;
    case 14: TIMESTAMP Timestamp;
    case 15: INFO_REQUEST InfoReply;
    case 16: INFO_REQUEST InfoRequest;
  }
} ICMP_MESSAGE;

entrypoint typedef struct _ICMP_PACKET {
  UINT8 Type;
  ICMP_MESSAGE(Type) icmp_message;
} ICMP_PACKET;
```

- Tests achieve **~93% line coverage** on Wireshark ICMP parser
- **Uncovered 1 bug in the handwritten *ICMP* 3D specification!**
- Semantically equivalent to the handwritten spec (after bug fix) using symbolic differential testing

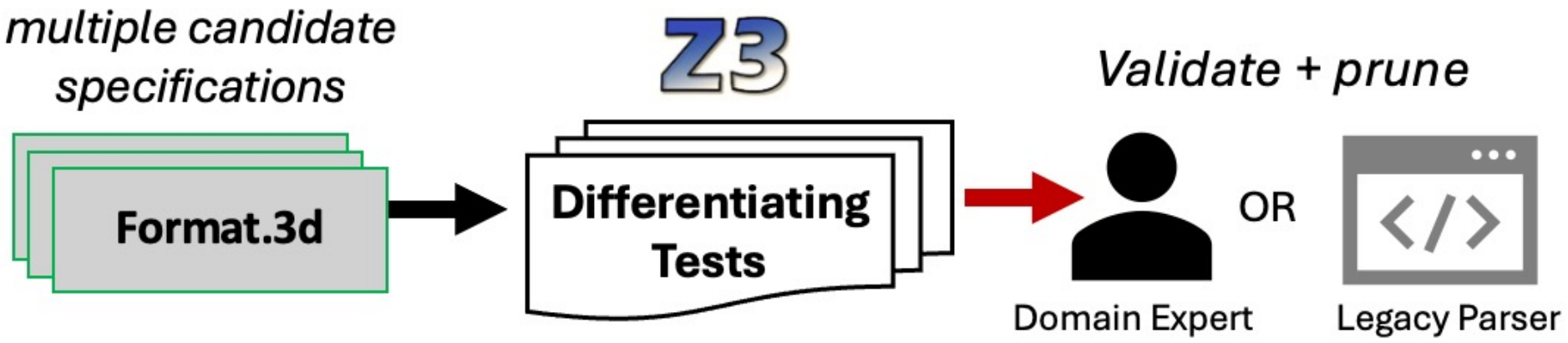
Choosing between multiple plausible candidates:

Multiple candidate specifications may **pass the test set**, but are **not *semantically equivalent***

1. Differentiate between candidate specifications using **symbolic test generation**
2. Validate differentiating tests against a **legacy parser**

What if I have no legacy parser?

Domain expert reviews **a set of differentiating tests** to identify desired behavior.



Microsoft

Protocol	Equivalent?	Root Cause Divergence	After H.S. Fix
UDP	✗	H.S. Missing constraint on Length field	✓
ICMP	✗	H.S. UNUSED_BYTES type too short	✓
VXLAN	✗	H.S. VXLanID field too short	✓
IPV6	✓	None	n/a
Ethernet	✓	None	n/a

Interpreting RFCs is not always easy

Writing specifications by hand is easy to get wrong!

Verifying internal Microsoft Parsers with 3DGen

Open-source

- 20 Standard Wireshark Protocols
- **Windows** eBPF IOCTLs, ELF file formats

Microsoft Internal

- **Azure Hyper-V** networking protocols
- **.NET Http.sys** IOCTLs

#	Protocol	RFC (Version)	Length (Pages)	Description
1	UDP*	768	3	User Datagram Protocol
2	ICMPv4 *	792	21	Internet Control Message Protocol
3	VXLAN*	7348	22	Virtual eXtensible Local Area Network
4	IPV6*	2460	39 (24)	Internet Protocol version 6
5	IPV4*	791	45 (12)	Internet Protocol version 4
6	TCP*	793	85 (10)	Transmission Control Protocol
7	Ethernet*	7348	22	Ethernet II Frames in VXLAN
8	GRE	2784	9	Generic Routing Encapsulation
9	IGMPv2	2236	24	Internet Group Managment Protocol
10	DHCP	2131	45 (4)	Dynamic Host Configuration Protocol
11	DCCP	4340	129 (14)	Datagram Congestion Control Protocol
12	ARP	826	10	Address Resolution Protocol
13	NTP	5905	110 (4)	Network Time Protocol
14	NBNS	1002	84 (6)	NetBIOS Name Service
15	NSH	8300	40 (8)	Network Service Header
16	TFTP	1350	11	Trivial File Transfer Protocol
17	RTP	3550	104 (3)	Transport Protocol for Real-Time Applications
18	PPP	1661	52 (11)	Point-to-Point Protocol
19	TPKT	2126	25	ISO Transport Service on top of TCP
20	OSPF	5340	94 (13)	Internet Official Protocol Standards

3DGen: AI-Assisted Generation of Provably Correct Binary Format Parsers

Sarah Fakhoury, Markus Kuppe, Shuvendu K. Lahiri, Tahina Ramananandro and Nikhil Swamy
Microsoft Research, Redmond, USA
{sfakhoury, makuppe, shuvendu, taramana, nswamy}@microsoft.com

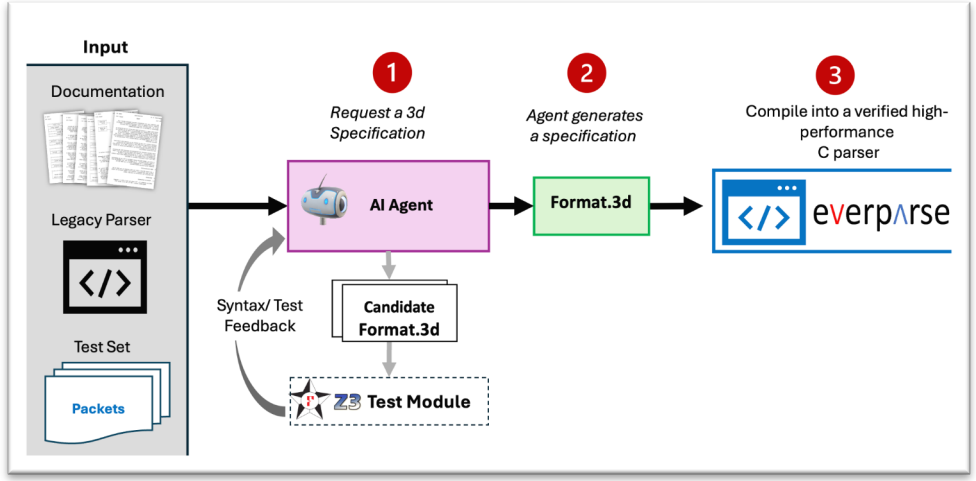
***Abstract*—Improper parsing of attacker-controlled input is a leading source of software security vulnerabilities, especially when programmers transcribe informal format descriptions in RFCs into efficient parsing logic in low-level, memory unsafe languages. Several researchers have proposed formal specification languages for data formats from which efficient code can be extracted. However, distilling informal requirements into formal specifications is challenging and, despite their benefits, new, formal languages are hard for people to learn and use.**

In this work, we present 3DGen, a framework that makes use of AI agents to transform mixed informal input, including natural language documents (i.e., RFCs) and example inputs into format specifications in a language called 3D. To support humans in understanding and trusting the generated specifications, 3DGen uses symbolic methods to also synthesize test inputs that can be validated against an external oracle. Symbolic test generation also helps in distinguishing multiple plausible solutions. Through a process of repeated refinement, 3DGen produces a 3D specification that conforms to a test suite, and which yields safe, efficient,

that yield trustworthy executable code. However, more commonly, specifications are not entirely formal and come from a variety of sources, ranging from natural language documents, diagrams, example code snippets, sample input/output pairs, etc. Extracting a formal specification from such a variety of sources requires a significant human effort, typically requiring a process that involves:

1) Learning a new DSL;
2) Understanding the informal specification;
3) Expressing one's understanding of the informal specification in the DSL;
4) Iterating to refine intent, revisiting the previous steps to arrive at a desired specification.

This is challenging enough that developers often directly transcribe informal specifications into executable code in general purpose programming languages, leaving the door open



Paper

Projects

- DSLs helpful intermediate abstractions for complex generation tasks
- DSLs unlock use of symbolic tools critical to guide LLMs + provide guarantees

More on Trusted AI-Assisted Programming at MSR!