Exploring Zero-Shot Prompting for Generating Data Format Descriptions

Prashant Anantharaman and Vishnupriya Varadharaju
Narf Industries

15th May 2025

Work performed as part of ARPA-H-DIGIHEALS program. Contract No. SP4701-23-C-0089

narf

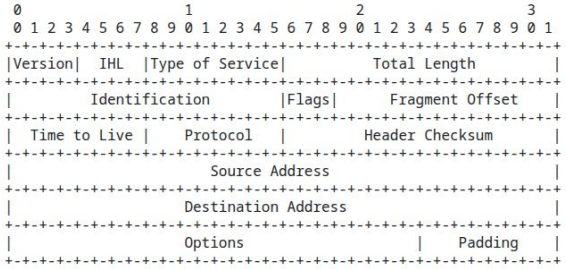
Building Parsers

- Instead of building a parser from scratch, all the constructs needed in parsers are available in tools

- **Parser Generators/Data Description Languages (DDLs):**
  - Provide tools to describe a programming language grammar or data format
  - Generate parsers in target languages (Java, C++, Python, etc.)
  - Parse input and provide an AST or Object containing the parsed data
  - DFDL also offers serialization

- **Parser Combinators:**
  - Create smaller parsers for individual structures and combine them into larger ones
  - Written directly in the target programming language — no need to learn an entirely new language
  - Tools like Hammer (bindings in several languages) and Nom (Rust-based) are suitable for parsing target binary data formats

DDL Syntax Examples



Hammer

```
h_sequence(
    ...
    h_uint16(), // total_length
    h_uint16(), // identification
    h_uint16(), // flags & offset
    h_uint8(),  // ttl
    h_uint8(),  // protocol
    h_uint16(), // header_checksum
    h_repeat_n(h_uint8(), 4), //
src_ip_addr
    h_repeat_n(h_uint8(), 4), //
dst_ip_addr
    NULL
);
```

Nom

```
...
let (input, length) =
number::streaming::be_u16(input)?;
let (input, id) =
number::streaming::be_u16(input)?;
let (input, flag_frag_offset) =
flag_frag_offset(input)?;
let (input, ttl) =
number::streaming::be_u8(input)?;
let (input, protocol) =
ip::protocol(input)?;
let (input, chksum) =
number::streaming::be_u16(input)?;
let (input, source_addr)= address(input)?;
let (input, dest_addr)=address(input)?;
...
```

DaeDaLus

```
def IPv4_header_s =
  struct
    ...
    total_length    : uint 16
    identification  : uint 16
    b67             : uint 16
    ttl             : uint 8
    protocol        : uint 8
    header_checksum : uint 16
    src_ip_addr     : [uint 8; 4]
    dst_ip_addr     : [uint 8; 4]
```

Derived from https://github.com/bestouff/pktparse-rs

DDL Syntax Examples (contd.)

Kaitai Struct

```
seq:
    ...
  - id: total_length
    type: u2be
  - id: identification
    type: u2be
  - id: b67
    type: u2be
  - id: ttl
    type: u1
  - id: protocol
    type: u1
  - id: header_checksum
    type: u2be
  - id: src_ip_addr
    size: 4
  - id: dst_ip_addr
    size: 4
```

DFDL

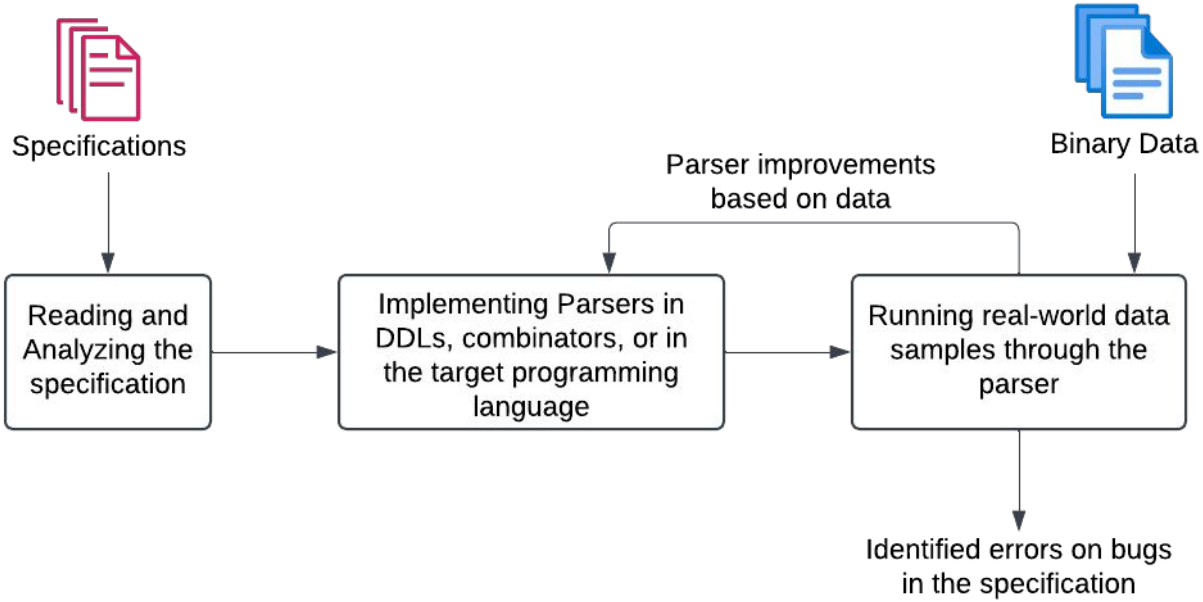
```
<xs:sequence>
    ...
    <xs:element name="Length"
type="b:bit" dfdl:length="16"/>
    <xs:element name="Identification"
type="b:bit" dfdl:length="16"/>
    <xs:element name="Flags"
type="b:bit" dfdl:length="3"/>
    <xs:element name="FragmentOffset"
type="b:bit" dfdl:length="13"/>
    <xs:element name="TTL" type="b:bit"
dfdl:length="8"/>
    <xs:element name="Protocol"
type="b:bit" dfdl:length="8"/>
    <xs:element name="Checksum"
type="chksum:IPv4Checksum"/>
    <xs:element name="IPSrc"
type="ip:IPAddress"/>
    <xs:element name="IPDest"
type="ip:IPAddress"/>
</xs:sequence>
```

Spicy

```
type ip4_hdr: record {
      ...
      len: count;
      id: count;
      DF: bool;
      MF: bool;
      offset: count;
      ttl: count;
      p: count;
      sum: count;
      src: addr;
      dst: addr;
};
```

Derived from:
https://github.com/DFDLSchemas/ethernetIP
https://github.com/zeek/zeek/blob/master/scripts/base/init-bare.zeek

Parser Methodology



Can LLMs automate some of these tasks?

Research Questions

RQ1: Can off-the-shelf LLMs produce DDL code that is *syntactically valid*?
H1: *All LLMs must be able to produce some sample specifications in each DDL syntax.*

RQ2: Does the generated DDL code cover 100% of the format specification?
H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs.*

RQ3: Does the generated DDL code reject malformed inputs?
H3: *LLM-generated specifications would accept several malformed inputs.*

RQ4: Can LLMs learn the syntax of DDLs they do not know using example specifications and manuals?
H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications.*

Research Questions

RQ1: Can off-the-shelf LLMs produce DDL code that is *syntactically valid*?
H1: *All LLMs must be able to produce some sample specifications in each DDL syntax.*

RQ2: Does the generated DDL code cover 100% of the format specification?
H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs.*

RQ3: Does the generated DDL code reject malformed inputs?
H3: *LLM-generated specifications would accept several malformed inputs.*

RQ4: Can LLMs learn the syntax of DDLs they do not know using example specifications and manuals?
H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications.*

Research Questions

RQ1: Can off-the-shelf LLMs produce DDL code that is *syntactically valid*?
H1: *All LLMs must be able to produce some sample specifications in each DDL syntax.*

RQ2: Does the generated DDL code cover 100% of the format specification?
H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs.*

RQ3: Does the generated DDL code reject malformed inputs?
H3: *LLM-generated specifications would accept several malformed inputs.*

RQ4: Can LLMs learn the syntax of DDLs they do not know using example specifications and manuals?
H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications.*

Research Questions

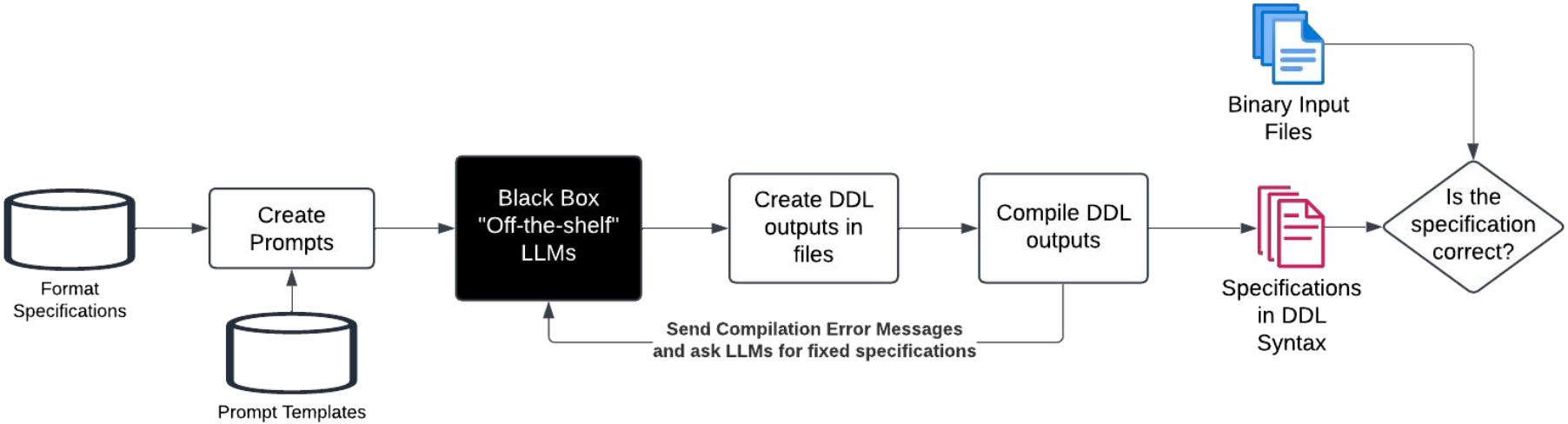
RQ1: Can off-the-shelf LLMs produce DDL code that is *syntactically valid*?
H1: *All LLMs must be able to produce some sample specifications in each DDL syntax.*

RQ2: Does the generated DDL code cover 100% of the format specification?
H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs.*

RQ3: Does the generated DDL code reject malformed inputs?
H3: *LLM-generated specifications would accept several malformed inputs.*

RQ4: Can LLMs learn the syntax of DDLs they do not know using example specifications and manuals?
H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications.*

Approach



- Five temperature settings between 0 and 1.
- Three retries for compilation errors where the error messages are fed back

Prompt Templates

First Prompt	You are a software developer who has read the {specification} for the {format}. Can you list all the fields in the specification along with all the values each field can take?
Generate Scripts	Can you use this knowledge to generate a {ddl} specification for the {format} in {output} format? Make sure to cover the entire specification, including any optional fields. Do not provide any text response other than the {format} specification. Show only the complete response. Do not wrap the response in any markdown.
Fixing Errors	The previous response gave me an error. Can you use this error message: "{message}" to improve the specification and give me an improved, complete, and fixed {ddl} specification in {output} format. Give me only the complete generated code and no text with it. Ensure that the previous requirements are still met.
From Samples	You are a software developer familiar with the {specification} for the {format}. Given various sample specifications for different formats in {output}, use the insights from these samples to generate a comprehensive {ddl} specification for {format} in {output} format. Ensure that the entire specification is covered, including all optional fields. Provide only the complete {format} specification without any additional text or markdown formatting. Here are sample specifications for reference {sample_text}.
Using Manuals	Study the attached documentation carefully for {ddl} to answer my upcoming questions.You are a software developer who has read the {specification} for the {format}. Can you use this knowledge and the {ddl} documentation shared previously to generate a {ddl} specification for the {format} in {output} format? Make sure to cover the entire specification, including any optional fields. Return the response containing only the specification without any explanation.

Experimental Setup

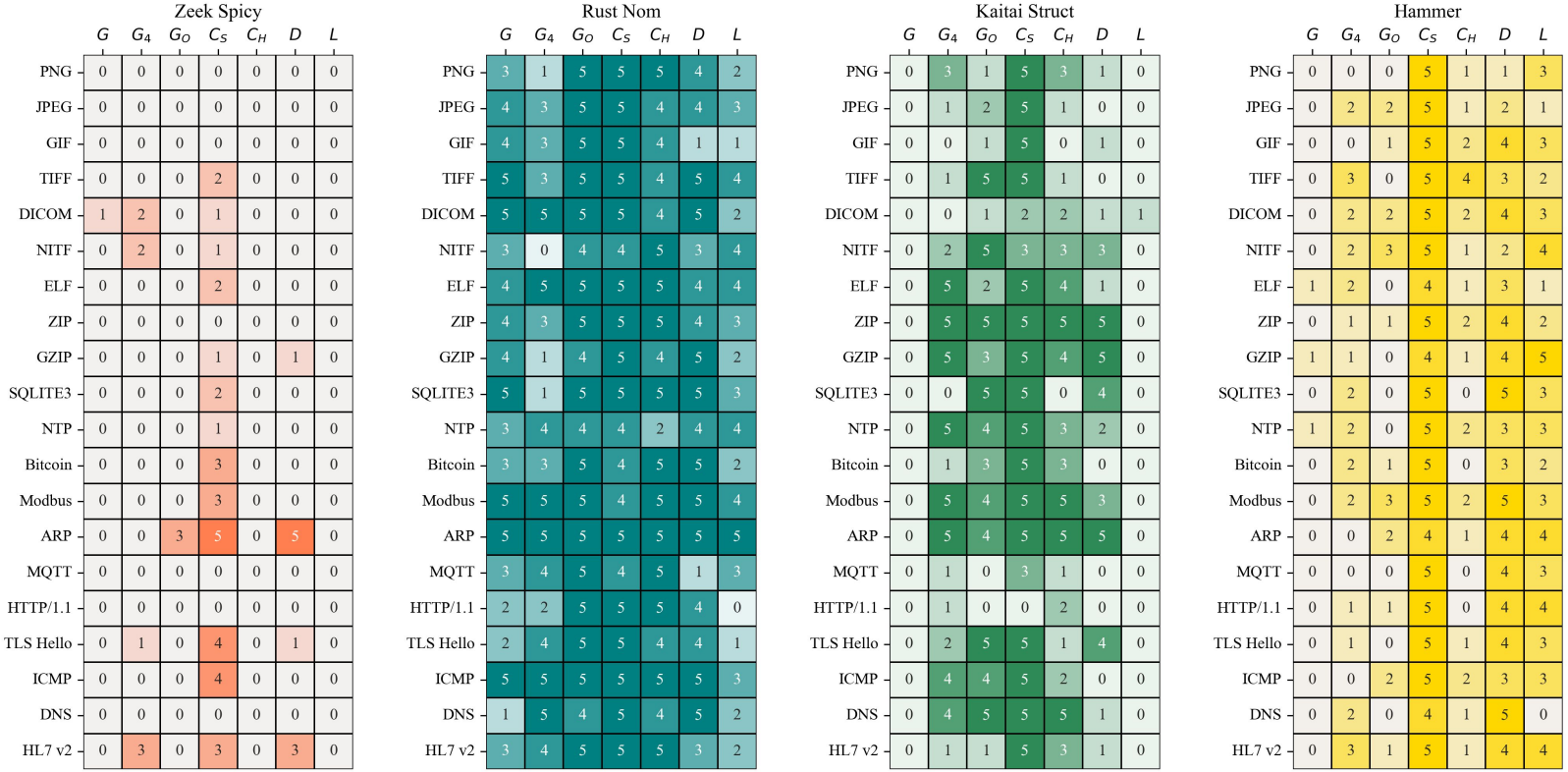
TABLE I
A COMPLETE LIST OF FORMATS, LLMS, AND DDLS WE INCLUDE IN OUR STUDY.

File Formats	Network Protocols	Parsing Technologies	LLM Models
PNG	DNS	Kaitai Struct	Claude 3.5 Sonnet
JPG	ICMP	DFDL	Claude 3.5 Haiku
GIF	Bitcoin Transactions	DaeDaLus	GPT-4o
TIFF	Modbus	Zeek Spicy	GPT-4-Turbo
NITF	NTP	Hammer	Llama-3-70B
DICOM	TLS Client Hello	Rust Nom	DeepSeek Coder V3
ELF	HTTP/1.1		Gemini 1.5 Flash
ZIP	MQTT		
GZIP	ARP		
SQLITE3	HL7 v2		

RQ1 Results

Do generated specifications compile?

Compilation Success



Compilation Success (contd.)

Daedalus

	G	G_4	G_O	C_S	C_H	D	L
PNG	0	0	0	0	0	0	0
JPEG	0	0	0	0	0	0	0
GIF	0	0	0	0	0	0	0
TIFF	0	0	0	0	0	0	0
DICOM	0	0	0	0	0	0	0
NITF	1	0	0	0	0	0	0
ELF	0	0	0	0	0	0	0
ZIP	0	0	0	0	0	0	0
GZIP	0	0	0	0	0	0	0
SQLITE3	0	0	0	0	0	0	0
NTP	0	0	0	0	0	0	0
Bitcoin	0	0	0	0	0	0	0
Modbus	0	0	0	0	0	0	0
ARP	0	0	0	0	0	0	0
MQTT	0	0	0	0	0	0	0
HTTP/1.1	0	0	0	0	0	0	0
TLS Hello	0	0	0	0	0	0	0
ICMP	0	0	0	0	0	0	0
DNS	0	0	0	0	0	0	0
HL7 v2	2	0	0	0	0	0	0

DFDL

	G	G_4	G_O	C_S	C_H	D	L
PNG	0	0	0	0	0	0	0
JPEG	0	0	0	0	0	0	0
GIF	0	0	0	0	0	0	0
TIFF	0	0	0	0	0	0	0
DICOM	0	0	0	0	0	0	0
NITF	0	0	0	0	0	0	0
ELF	0	0	0	0	0	0	0
ZIP	0	0	0	0	0	0	0
GZIP	0	0	0	0	0	0	0
SQLITE3	0	0	0	0	0	0	0
NTP	0	0	0	0	0	0	0
Bitcoin	0	0	0	0	0	0	0
Modbus	0	0	0	0	0	0	0
ARP	0	0	0	0	0	0	0
MQTT	0	0	0	0	0	0	0
HTTP/1.1	0	0	0	0	0	0	0
TLS Hello	0	0	0	0	0	0	0
ICMP	0	0	0	0	0	0	0
DNS	0	0	0	0	0	0	0
HL7 v2	0	0	0	0	0	0	0

Deep Dive into Generated Parsers

Hammer and Llama

Llama removes the entire library and produces a struct and C parser from scratch

```
typedef struct {
    uint16_t htype;
    uint16_t ptype;
    uint8_t hlen;
    uint8_t plen;
    uint16_t oper;
} arp_header_t;
```

DaeDaLus and Gemini

```
# This is an incomplete
placeholder. A full DICOM
specification is impossible in
Daedalus.
```

RQ2 and RQ3 Results

Evaluating the correctness of the compiled specifications

RQ2 and RQ3: Validating Specification Accuracy

Datasets:

- GovDocs1 for Image Files: PNG (4,125), JPEG (109,283), and GIF (36,302)
- Wireshark GitHub Repository for NTP files
- New York Power Authority's Lab for Modbus packets

Invalid Files: Mutated each file at least three times to produce invalid files using the Fuzzing Book Mutation fuzzer.

Ground Truth: Used the Pillow Library for Image files and Wireshark for network packets

Precision: (Accepted valid inputs/Total accepted inputs)

Recall: (Accepted valid inputs/Total valid inputs)

Parser Correctness: Precision and Recall of 1.0

Precision

Formats	Temp.	Kaitai Struct							Hammer							Rust Nom						
		G	G_4	G_O	C_S	C_H	D	L	G	G_4	G_O	C_S	C_H	D	L	G	G_4	G_O	C_S	C_H	D	L
JPEG	**0.0**	-[a]	N/A[b]	-	N/A	-	-	-	-	0.45	-	N/A	-	N/A	0.42	0.42	0.42	0.42	0.42	-	0.42	0.00[c]
	0.25	-	N/A	N/A	N/A	-	-	-	-	-	0.42	N/A	-	-	-	0.42	-	0.42	0.40	N/A	-	0.44
	0.5	-	N/A	N/A	N/A	N/A	-	-	-	0.42	-	N/A	-	-	-	-	0.42	0.42	0.42	0.45	0.42	N/A
	0.75	-	N/A	N/A	N/A	N/A	-	-	-	-	N/A	N/A	N/A	N/A	-	0.42	-	0.42	0.42	0.42	0.42	-
	1.0	-	N/A	N/A	N/A	N/A	-	-	-	-	-	N/A	-	-	-	0.42	0.42	0.42	0.42	N/A	0.42	-
GIF	**0.0**	-	-	N/A	1.00[d]	-	-	-	-	-	-	0.66	-	-	N/A	0.50	-	0.50	0.50	N/A	0.50	0.50
	0.25	-	-	N/A	1.00	-	-	-	-	-	-	0.66	-	0.66	-	0.50	-	0.50	0.50	0.50	-	-
	0.5	-	-	N/A	1.00	-	-	-	-	-	-	0.66	N/A	0.66	0.50	0.56	0.50	0.50	0.50	N/A	-	-
	0.75	-	-	N/A	1.00	-	1.00	-	-	-	0.00	1.00	N/A	0.65	-	-	0.50	0.50	0.50	-	-	-
	1.0	-	-	N/A	1.00	-	1.00	-	-	-	-	0.66	-	1.00	N/A	0.50	0.51	0.50	0.95	1.00	-	-
Modbus	**0.0**	-	1.00	N/A	0.02	1.00	N/A	-	-	-	N/A	1.00	-	1.00	-	0.74	0.74	0.74	1.00	N/A	0.74	0.92
	0.25	-	1.00	N/A	0.02	1.00	N/A	-	-	-	0.90	1.00	1.00	1.00	-	0.74	0.74	0.81	0.74	N/A	0.74	-
	0.5	-	1.00	N/A	0.02	1.00	N/A	-	-	0.74	-	1.00	-	0.74	0.74	0.74	0.81	1.00	0.74	N/A	0.74	0.91
	0.75	-	1.00	N/A	0.02	1.00	N/A	-	-	1.00	-	1.00	-	N/A	0.74	0.74	0.74	0.74	0.74	1.00	0.74	0.74
	1.0	-	1.00	N/A	0.02	1.00	N/A	-	-	-	1.00	1.00	N/A	N/A	0.74	0.74	0.81	0.74	-	0.92	0.74	0.75
ARP	**0.0**	-	-	-	1.00	1.00	1.00	-	-	-	1.00	0.33	-	0.33	0.38	0.33	0.33	0.33	1.00	1.00	0.33	0.33
	0.25	-	-	-	1.00	1.00	1.00	-	-	-	-	0.38	-	0.38	0.33	0.33	0.33	0.33	1.00	1.00	0.33	0.33
	0.5	-	-	-	1.00	1.00	1.00	-	-	-	-	0.38	-	N/A	-	0.33	0.33	0.33	1.00	1.00	0.33	0.33
	0.75	-	-	-	1.00	1.00	1.00	-	-	-	-	-	-	-	0.33	0.33	0.33	0.33	1.00	1.00	0.33	0.33
	1.0	-	-	-	1.00	1.00	1.00	-	-	-	0.38	1.00	N/A	0.38	0.33	1.00	0.33	0.33	1.00	0.38	0.33	0.38

[a] –: denotes cases that did not compile.
[b] `N/A`: denotes cases that did not produce any successful parses.
[c] `0.00`: denotes cases where valid files were not parsed, but files we categorized as invalid parsed successfully.
[d] `1.00`: denotes cases where we did not parse any invalid files.

Recall

Formats	Temp.	Kaitai Struct							Hammer							Rust Nom						
		G	G_4	G_O	C_S	C_H	D	L	G	G_4	G_O	C_S	C_H	D	L	G	G_4	G_O	C_S	C_H	D	L
JPEG	**0.0**	-[a]	0.00[b]	-	0.00	-	-	-	-	0.37	-	0.00	-	0.00	1.00[c]	1.00	1.00	0.94	1.00	-	1.00	0.00
	0.25	-	0.00	0.00	0.00	-	-	-	-	-	1.00	0.00	-	-	-	1.00	-	0.94	0.74	0.00	-	1.00
	0.5	-	0.00	0.00	0.00	0.00	-	-	-	1.00	-	0.00	-	-	-	-	0.95	0.94	1.00	1.00	1.00	0.00
	0.75	-	0.00	0.00	0.00	0.00	-	-	-	-	0.00	0.00	0.00	0.00	-	1.00	-	0.94	1.00	1.00	1.00	-
	1.0	-	0.00	0.00	0.00	0.00	-	-	-	-	-	0.00	-	-	-	1.00	1.00	1.00	1.00	0.00	1.00	-
GIF	**0.0**	-	-	0.00	1.00	-	-	-	-	-	-	0.34	-	-	0.00	1.00	-	1.00	1.00	0.00	1.00	1.00
	0.25	-	-	0.00	1.00	-	-	-	-	-	-	0.34	-	0.34	-	1.00	-	1.00	0.99	1.00	-	-
	0.5	-	-	0.00	1.00	-	-	-	-	-	-	0.34	0.00	0.34	1.00	0.97	1.00	1.00	1.00	0.00	-	-
	0.75	-	-	0.00	1.00	-	0.11	-	-	-	0.00	0.00	0.00	0.34	-	-	1.00	1.00	1.00	-	-	-
	1.0	-	-	0.00	1.00	-	0.11	-	-	-	-	0.34	-	0.01	0.00	0.99	1.00	1.00	0.70	0.00	-	-
Modbus	**0.0**	-	0.00	0.00	0.00	0.00	0.00	-	-	-	0.00	0.50	-	0.67	-	1.00	1.00	1.00	0.33	0.00	1.00	0.43
	0.25	-	0.00	0.00	0.00	0.00	0.00	-	-	-	0.67	0.67	0.67	0.67	-	1.00	1.00	1.00	1.00	0.00	1.00	-
	0.5	-	0.00	0.00	0.00	0.00	0.00	-	-	1.00	-	0.67	-	1.00	1.00	1.00	1.00	0.67	1.00	0.00	1.00	0.41
	0.75	-	0.00	0.00	0.00	0.00	0.00	-	-	0.67	-	0.33	-	0.00	1.00	1.00	1.00	1.00	1.00	0.67	1.00	1.00
	1.0	-	0.00	0.00	0.00	0.00	0.00	-	-	-	0.67	0.50	0.00	0.00	1.00	1.00	1.00	1.00	-	0.40	1.00	0.47
ARP	**0.0**	-	-	-	1.00	1.00	1.00	-	-	-	0.15	1.00	-	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	0.25	-	-	-	1.00	1.00	1.00	-	-	-	-	1.00	-	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	0.5	-	-	-	1.00	1.00	1.00	-	-	-	-	1.00	-	0.00	-	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	0.75	-	-	-	1.00	1.00	1.00	-	-	-	-	-	-	-	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	1.0	-	-	-	1.00	1.00	1.00	-	-	-	1.00	0.15	0.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

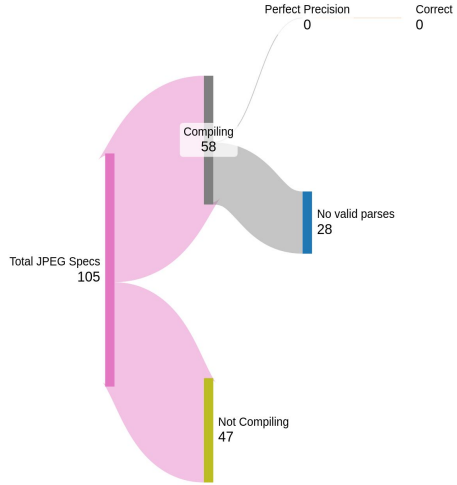
[a] –: denotes cases that did not compile.

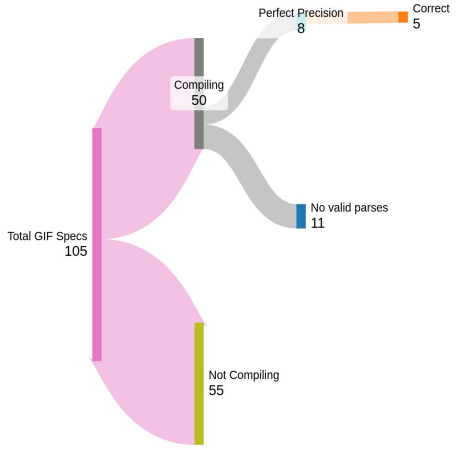
[b] `0.00`: denotes cases where valid files were not parsed.

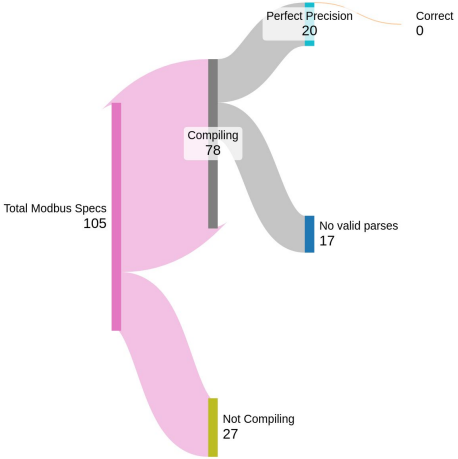
[c] `1.00`: denotes cases where we did not incorrectly mark any valid files as invalid.

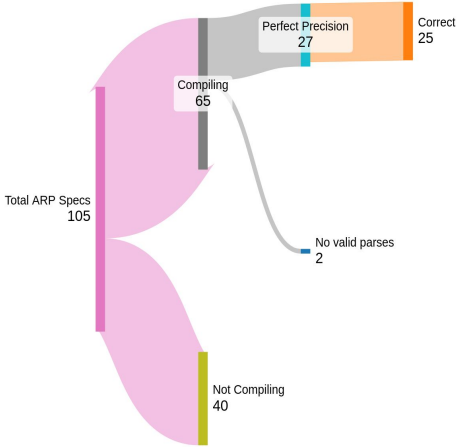
RQ2 and RQ3 Summary

- Totally 140 generations in Kaitai, Hammer, and Nom each
  - 31 Kaitai, 19 Hammer, and 9 Nom implementations accepted no inputs (valid or invalid) at all
- Observed 30 cases of perfect precision and recall, but none in Hammer
  - 5 GIF implementations produced by Claude Sonnet in Kaitai Struct syntax
  - 15 ARP implementations produced by Claude Sonnet, Haiku, and Deepseek in Kaitai Struct
  - 10 ARP implementations produced by Claude Sonnet and Haiku in Rust Nom syntax
- None of the JPEG and Modbus implementations met the correctness criteria set by us









RQ4 Results

Providing sample specifications and manuals

RQ4: Providing sample specifications and the manual

- Manuals are often PDFs or web pages. LLMs were not able to understand much after PDF-to-text operations, and the manuals were usually too large to upload as is to web interfaces.
    - We studied DaeDaLus and Spicy since they had shorter manuals.
    - The manuals did not produce any improvements in the performance of the LLMs at producing specifications, with identical results to zero-shot prompting
- Fed 5 sample specifications each for Spicy and DaeDaLus, and found an improvement in Spicy specifications for JPEG and Modbus implementations.

Zeek Spicy without samples

	G	G_4	G_O	C_S	C_H	D	L
JPEG	0	0	0	0	0	0	0
Modbus	0	0	0	3	0	0	0

Zeek Spicy

	G	G_4	G_O	C_S	C_H	D	L
JPEG	0	0	0	1	0	3	1
Modbus	0	0	4	3	2	3	0

Hypotheses Revisited

H1: *All LLMs must be able to produce some sample specifications in each DDL syntax.*
Result: False; no valid specifications in DaeDaLus and DFDL syntax.

H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs.*
Result: True

H3: *LLM-generated specifications would accept several malformed inputs.*
Result: True

H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications.*
Result: Partially True; manuals did not help, but examples did.

Hypotheses Revisited

H1: *All LLMs must be able to produce some sample specifications in each DDL syntax.*
Result: False; no valid specifications in DaeDaLus and DFDL syntax.

H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs.*
Result: True

H3: *LLM-generated specifications would accept several malformed inputs.*
Result: True

H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications.*
Result: Partially True; manuals did not help, but examples did.

Hypotheses Revisited

H1: *All LLMs must be able to produce some sample specifications in each DDL syntax.*
Result: False; no valid specifications in DaeDaLus and DFDL syntax.

H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs.*
Result: True

H3: *LLM-generated specifications would accept several malformed inputs.*
Result: True

H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications.*
Result: Partially True; manuals did not help, but examples did.

Hypotheses Revisited

H1: *All LLMs must be able to produce some sample specifications in each DDL syntax.*
Result: False; no valid specifications in DaeDaLus and DFDL syntax.

H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs.*
Result: True

H3: *LLM-generated specifications would accept several malformed inputs.*
Result: True

H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications.*
Result: Partially True; manuals did not help, but examples did.

Conclusions

- **Hammer, Nom, and Kaitai Struct** seemed to have the most compiling specifications, with Nom and Kaitai having more *correct* specifications. They might be appropriate candidates for LLM-assisted generation in the future.

- A large portion of the specifications we generated did not compile, and among the ones that did a very small subset turned out to be correct.

- LLMs are not very successful at generating DDL code when there are not many sample specifications available on the Internet (DaeDaLus and Spicy).

Future Directions

- Patching specifications based on failing inputs
- Using newer reasoning engines to conduct a similar study
- Performing translations from one DDL syntax to another
- Systematically exploring what file format features and corresponding DDL features prevent accurate LLM-generated specifications

Thank You

https://github.com/narfindustries/llm-tests-langsec
https://prashant.at/files/llm-langsec25.pdf

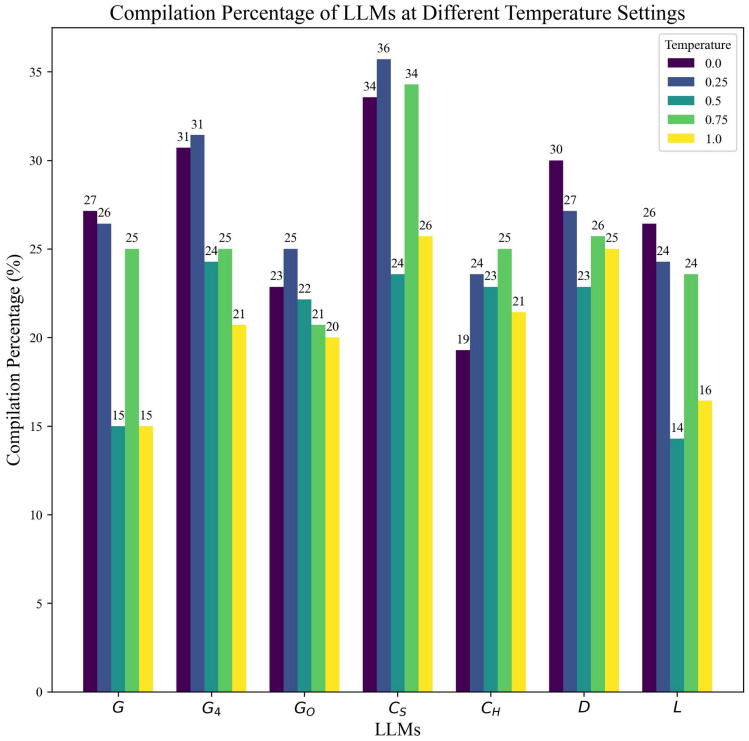
prashant.anantharaman@narfindustries.com

Some other discussion items

- **Postel's Robustness Principle and permissiveness:** how do you evaluate ground truth with Wireshark and Pillow knowing they are permissive? Differential fuzzing of any generated parser to existing real-world parsers would be the best way forward.

- **Reproducibility Challenges:** *It is challenging….*
  We released our source code and prompt templates and set the model versions.

Does Temperature Affect Code Generation?



- Claude 3.5 Sonnet and GPT-4-Turbo comparatively produce far more compilable specifications
- However, temperature variations do not show consistent effects on compilation success

Hypotheses

RQ1: Can off-the-shelf LLMs produce DDL code that is *syntactically valid*?
H1: *All LLMs must be able to produce some sample specifications in each DDL syntax*

RQ2: Does the generated DDL code cover 100% of the format specification?
H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs*

RQ3: Does the generated DDL code reject malformed inputs?
H3: *LLM-generated specifications would accept several malformed inputs.*

RQ4: Can LLMs learn the syntax of DDLs they do not know using example specifications and manuals?
H4: *Being provided with specifications and manuals, LLMs should be able to provide compilable DDL specifications*

Research Questions

RQ1: Can off-the-shelf LLMs produce DDL code that is *syntactically valid*?

RQ2: Does the generated DDL code cover 100% of the format specification?

RQ3: Does the generated DDL code reject malformed inputs?

RQ4: Can LLMs learn the syntax of DDLs they do not know using example specifications and manuals?

The Core Problem

Parser Vulnerabilities

Paragon and Pegasus attacks:

- FORCEDENTRY: Vulnerabilities in the JBIG2 image parsing library
- libwebp Vulnerabilities: Vulnerabilities in image parsing libraries used in Chromium-based browsers
- LogoFAIL: Vulnerabilities in image-parsing libraries of AMI, Insyde, and Phoenix BIOS meant to parse personalized BIOS logos

Parser Differentials

- PDF Attacks where two readers show different content
- X.509 certificates: different parsers disagree on who was granted the certificate
- HTTP Request Smuggling: Bypassing some security protections/guarantees on HTTP proxies/middleboxes

Hypotheses Revisited

H1: *All LLMs must be able to produce some sample specifications in each DDL syntax*
Result: False; no valid specifications in DaeDaLus and DFDL syntax

H2: *LLMs tend to miss portions of the specifications, and rarely accept all valid sample inputs*
Result: True

H3: *LLM-generated specifications would accept several malformed inputs.*
Result: True

H4: *Being provided with example specifications and manuals, LLMs should be able to provide compilable DDL specifications*
Result: Partially True; manuals did not help, but examples did.