PROSPER: Extracting Protocol Specifications Using Large Language Models

Prakhar Sharma
SRI International
prakhar.sharma@sri.com

Vinod Yegneswaran
SRI International
vinod@csl.sri.com

Abstract

We explore the application of Large Language Models (LLMs) (specifically GPT-3.5-turbo) to extract specifications and automating understanding of networking protocols from Internet Request for Comments (RFC) documents. LLMs have proven successful in specialized domains like medical and legal text understanding, and this work investigates their potential in automatically comprehending RFCs. We develop Artifact Miner, a tool to extract diagram artifacts from RFCs. We then couple extracted artifacts with natural language text to extract protocol automata using GPT-turbo 3.5 (chatGPT) and present our zero-shot and few-shot extraction results. We call this framework for FSM extraction 'PROSPER: Protocol Specification Miner'. We compare PROSPER with existing state-of-the-art techniques for protocol FSM state and transition extraction. Our experiments indicate that employing artifacts along with text for extraction can lead to lower false positives and better accuracy for both extracted states and transitions. Finally, we discuss efficient prompt engineering techniques, the errors we encountered, and pitfalls of using LLMs for knowledge extraction from specialized domains such as RFC documents.

CCS Concepts

- Networks → Network protocol design; Protocol testing and verification; - Information systems → Information extraction; Summarization;

Keywords

Large language models, request for comments, protocol specifications, protocol FSMs, automated extraction

ACM Reference Format:

1 Introduction

Network protocols serve as the foundation for communication between devices and systems but often are complex and diverse, making manual analysis and implementation time-consuming and error-prone. A common way of specifying network protocols is using request for comments (RFC) documents. Automatic protocol understanding from RFCs has numerous applications viz. attack synthesis [24] and protocol security analysis [17], generation of implementation guidelines (in conjunction with domain experts) [1], network troubleshooting [20], and code de-bloating [21]. Extracting a formal model of a protocol from RFCs via natural language processing (NLP) is a challenging task for many reasons. First, RFCs contain protocol definitions in natural text which is inherently ambiguous. Second, a canonical finite state machine (FSM) specification is based not only on the information contained in the RFCs, but also on inputs from domain experts. Third, deep-learning-based information extraction and semantic parsing [14] has shown some promising results on benchmark datasets like wikitableQuestions [19] among others, but require both high quality and high volume of annotated data. When dealing with technical domains like network protocols, generating high quality annotated data is a tough and expensive endeavor. Finally, all the elements of the canonical FSM present in the RFC are not exhaustively covered in text. A key element of RFCs is textual diagrams, which are pictorial depictions of key protocol elements like data flow procedures, connection procedures, message structures, packet headers, variable definitions etc.; and often contain information that is not explicitly mentioned in text.

The case for foundational NLP models

Large Language Models [2, 8, 11, 16, 22] (LLMs) are a groundbreaking advancement in natural language processing (NLP) that have revolutionized the field of artificial intelligence. These models, such as Generative Pre-trained Transformer (GPT) [16], are characterized by their massive size, extensive pretraining on diverse text data, and impressive language generation capabilities. LLMs have billions of parameters, enabling them to capture intricate language patterns and understand complex semantic relationships. Hence, LLMs have been applied to a wide range of applications, from chatbots and content creation to translation and code generation. They are increasingly being applied to specialized domains like medical and legal language understanding. Given the generative capabilities of these models, and the aforementioned challenges in protocol information extraction, we posit the following
research questions (RQs) that we will attempt to answer through this paper:

- **RQ1**: How do foundational models, specifically chatGPT, compare against other state-of-the-art techniques for protocol information extraction (effectiveness)?
- **RQ2**: How does the effectiveness of LLM-based extraction vary across RFCs (generalizability)?
- **RQ3**: Can we utilize the non-textual components of an RFC to augment information derived purely using natural language (coverage enrichment)?

**Prior Work.** Previous efforts to apply NLP techniques for automating network protocol understanding (e.g., HYPER [18] and DASE [27]) use semantic parsing to extract information from man pages, documentations, and source code. Others like Veritas [25] and Prospex [6] analyse network trace messages to automatically generate protocol automata. The former extracts a probabilistic protocol state machine, while the latter uses network message clustering. Cho et al. [4] use a set of end-user provided abstraction functions to generate and abstract alphabets out of trace messages. MACE [5] uses manual extraction of message formats from the RFC and then symbolic execution for protocol verification. FSMGen [13] employs program analysis to extract state machines from TinyOS programs. Bander [7] infer the protocol state machine directly from Java source code, while Lie et al. [15] used an extensible compiler system, xgrep+, to perform protocol extraction.

More recently, SAGE [29] uses NLP to convert RFCs to an intermediate logical form, that it later uses for protocol disambiguation and code generation in a semi-automated fashion. Similarly, RFCNLP [17] uses a pre-trained BERT model fine tuned on protocol specification documents. It defines a Backus-Naur form grammar for pre-defined protocol terms viz. source states, destination states, transitions, triggers (events) etc. and tags RFC documents using those tags. It then extracts protocol FSMs using promela (Process Meta-Language).

To the best of our knowledge, RFCNLP and our approach are the only two that use Transformer-based [23] architectures to extract protocol FSMs from protocol specification documents. Most of the aforementioned works rely heavily on either human input, or manual extraction from protocol definitions, or on availability of protocol source code. Some of the drawbacks of these approaches include: (i) They aim to extract only a partial (or probabilistic) FSM; (ii) The approach is tailored to work well with a specific format of RFCs like transport layer protocols or IP protocols; (iii) The rigid grammar defined for RFC tagging only works with a subset of protocols, e.g., transport-layer protocols and might fail to extract meaningful transitions in application-layer protocols. To address these issues, we present PROSPER: A general approach to extracting protocol specifications from RFCs.

**Contributions.** The contributions of our work include:

- Development of PROSPER, a framework to extract finite state machine transitions from RFCs. PROSPER brings together natural language text and textual artifacts to achieve state-of-the-art accuracy in automatic FSM extraction using LLMs.
- Development of Artifact Miner, a simple yet powerful approach to extract non-textual artifacts from RFCs.
- Application of a host of techniques (topic modeling, union-find, TF-IDF [9], expert input) to select a set of representative RFCs for prompt engineering and extraction validation.
- Discussion of LLM prompt engineering approaches for the RFC based FSM extraction task and the benefits of using artifacts along with natural language with exemplar RFCs.
- Presentation of evaluation results and discussion of strategies for zero-shot FSM extraction and artifact understanding. When compared to prior work, PROSPER achieves more coverage and is more generalizable to a diverse set of RFCs.

## 2 System Design

### 2.1 RFC Selection Methodology

During the RFC selection process, careful consideration was given to ensure representative coverage of various important networking protocols that form the backbone of the current networking and communication infrastructure. We draw comparisons of our approach with RFCNLP [17], hence the RFCs that make for the bulk of their experiments (DCCP, PPTP and TCP in particular) are also part of the final RFC dataset that we worked with. A preliminary filtering process involved three steps:

- A BERT based topic model [10] that clusters RFCs into different topics. Emphasis was laid on selected RFCs from a diverse set of topics.
- Each RFC reports other RFCs that it obsoletes, updates, or is updated by. We converted this information into an adjacency list and subsequently into a graph of connected components. We chose one RFC from each of these ‘islands’ of connected components.
- Specific RFCs that are highly regarded or influential in the networking domain were also prioritized in the selection process.

It should be noted that the strategies mentioned above were only employed as guidance, instead of a deterministic algorithm in the selection process. The selected set of RFCs were split into two sets: train and validation. The training set was used for prompt engineering. We present our results, and drill down on specific corner-cases for a subset of these RFCs in Section 3.

### 2.2 RFC Cleansing

RFCs are complicated documents that do not adhere to a strict design or template; hence the RFC cleaning process is particularly challenging. We developed the following general rules for cleaning the selected RFCs:

- All RFC headers containing author names, page numbers, publication year information, and track information were removed because they do not contribute to protocol definition.
- Our experiments showed that including the table of contents in the LLM prompting process led the models to try
We approach protocol information extraction using LLMs, which was done to adhere to maximum context lengths of GPT3.5-turbo. Variables defined in RFC specifications that form parts of textual diagrams (large context) present extraneous information for the LLM to generate a better answer for benchmark tasks [30], we also observe that true that a larger input context helps the LLM to generate a more granular extraction of information.

2.3 RFC Chunking
Cleaned RFCs were split into chunks of 500 lines each. The maximum characters in a line in all the RFCs (including the representative RFCs selected above) is 82. Hence each chunk consisted of a maximum of 410,000 characters. This was done to adhere to maximum context lengths of GPT3.5-turbo. Note that the chunks contained both the plain text and textual figures. As demonstrated by experiments in this paper, LLMs (particularly GPT3.5-turbo) are good at reading and extracting information from both the textual concepts, and the figures (which we refer to as artifacts); hence, we chose to leave that information in the state machine extraction prompts. We also perform artifact only experiments for a more granular extraction of information.

2.4 Automating RFC Protocol Understanding
We approach protocol information extraction using LLMs from two different perspectives: automatically extracting FSMs from protocol definitions, and understanding the structure of control messages as defined in RFCs as such information is valuable for various networking problems like traffic analysis, intrusion detection etc.

Extracting FSMs from Natural Language specifications. Finite state machines are defined by three key entities: the source state that a system can be in, the destination state that the system transitions to, and the trigger (event) that causes the transition. We derive motivation from previous works [17] that defined a finite state machine grammar, and tagged RFCs in an xml like fashion; but instead of using the full grammar specified in [17], we posit that given a section of the RFC in natural language, the LLM will decipher the correct entities. For example, we just query the LLM for states and transitions instead of ‘source states’ and ‘destination states’ and associated transitions.

Extracting state variable and packet header description from textual diagrams. There are several kinds of variables defined in RFC specifications that form parts of packets that are sent out while initiating a connection, or stored locally and incremented based on some signal that is received. Most of the variables that are used in a protocol specification are explained as textual diagrams along with their descriptions in text. Our experiments show that GPT3.5-turbo can understand and extract information from these textual diagrams better than natural language text for the same task. We list several benefits of using these diagrams as prompt inputs instead of natural language text: (i) While it is true that a larger input context helps the LLM to generate a better answer for benchmark tasks [30], we also observe that for the purpose of information extraction, a large body of text (large context) presents extraneous information for the LLM that can lead to false positives (ii) RFC textual diagrams succinctly pack information that is seldom also mentioned in text. Some examples of this from RFC793 (TCP) and RFC2637 (PPTP) are provided in Section 3 (evaluation).

Extracting information from textual artifacts using Artifact Miner. We follow a two-step recipe to extract information from artifacts present in protocol specification documents. First, we develop artifact extractors to extract textual diagrams from the RFCs. Second, we include the extracted artifacts in the engineered prompt and feed it to the LLM. We discuss the design of the artifact extractor in the next section. The ‘lower arm’ in Figure 1 illustrates the overall design of the artifact mining process. Artifact Miner extracts textual diagrams using three broad techniques: Heuristic (symbol) based, multi-layer perceptron (MLP) based, and sequence model based extraction. In this paper we discuss the heuristic-based method. We observed that pictorial artifacts in an RFC like diagrams, topologies, call flows and message structures have key-symbols (’#’, ’[’, ’\’, ’...’) that occur in lines that correspond to that artifact. Heuristic-based extraction follows Algorithm 1. We then prune each artifact to make sure the extracted entities are indeed true artifacts. Although this extraction strategy is semi-manual in nature, and prone to errors, e.g., a line might contain a symbol even though it is not part of an artifact, the qualitative results on a majority of RFCs are surprisingly good; and since we have a fixed set of symbols, the algorithm scales linearly with the number of lines in the RFC.

Engineering LLM prompts: We split our representative RFCs into two sets: train and validation. We start with a seed prompt, which is a naively designed prompt that has the task description but is not optimized. These seed prompts are improved using the RFCs from the train set. We follow two broad strategies for prompt engineering: strategy 1: manual (greedy) and strategy 2: automatic prompt engineering, inspired by APE [30]. Figure 3 illustrates the seed prompt and the greedy version of the prompt that was finally used to extract the FSMs from RFC chunks. For strategy 1, we used the DCCP and TCP as train RFCs for prompt engineering. For strategy 2, we used DCCP and BGP as train RFCs. We

---

### Algorithm 1 Heuristic-based RFC artifact extraction

```
Require: Cleaned representative RFC
1: l ← [s, s, →, ...]
2: r ← cleaned RFC, artifact_set ← empty_set
3: for line in r do
4:    line_index ← current_line_index
5:    for symbol in 1 do
6:        if s is in line then
7:            idx ← line_index
8:            while line != "n" do
9:                idx ← 1
10:               top_idx ← idx
11:               idx ← line_index
12:               while line != "n" do
13:                    idx+ = 1
14:               bottom_idx ← idx
15:               Add r[top_idx : bottom_idx] to artifacts_set
16:               break
```

---

and guess the control statements that might be present in the RFC. To mitigate such false Positives, the table of contents were removed.
- The references and appendices, along with spurious new-lines and white-space characters were also removed.
used Algorithm 1 presented in [30] with the forward generation template, and employed a 0-1 scoring function that evaluated the prompt proposals by counting the correctly extracted transitions.

### 3 System Evaluation

We discuss some qualitative benefits and present experimental results that address the 3 RQs from Section 1: effectiveness, generalizability and coverage enrichment.

#### 3.1 Qualitative benefits of using pre-trained LLMs

Extracting perfect information from RFCs is a challenging task. Canonical FSMs are created based not only on RFCs but also on input from experts with exposure to protocol implementations, and often also rely on analyzing the code and textual diagrams present in the RFC, along with plain text. This situation is exacerbated by missing/ambiguous text in RFCs. Our experiments suggest that leveraging LLMs provide the following benefits over heuristic-based, or even fine-tuned transformer-based language models:

- **LLMs can self evaluate (RQ1: effectiveness)** A notable difference between our approach and xml-like semantic tagging in [17] is the non-determinism of LLM output. Asking the LLM vaguely to return entities in a specific format could range the outputs from a list of lists, to a hashmap of lists and so on. To circumvent this problem, we query the LLM for python code that uses the previous LLM output to draw FSMs using the ‘pygraphviz’ package. Examples of extracted FSMs are in Figure 2.

- **Pre-trained LLMs can generalize to most RFCs (RQ2: generalizability)** We make this argument based on our experiments, as we chose RFCs from a diverse set of domains. Previous approaches to automatic FSM extraction have focused on converting the plain text RFC to an RFC agnostic intermediate form, which is then converted to an approximate FSM. This process depends on the FSM grammar employed in generating the intermediate representation. [17] cites the challenge of coming up with an exhaustive grammar that fits most RFCs as one of the limitations of their approach. Since foundational models are trained on a considerable chunk of the internet [2], which includes technical forums, blogs, research papers, and specification documents, they arguably understand most RFC formats. While we realize that fine tuning these models even further might lead to better performance, we leave that as future work.

- **LLMs can understand beyond plain text (RQ3: coverage enrichment).** RFCs are complex technical documents where a lot of information pertinent to the protocol is expressed as textual artifacts. These textual artifacts employ characters to represent complicated connected shapes and to express protocol entities, e.g. transitions, states, packet headers, communication flows, data flow diagrams, message structures etc. and this results in ambiguous text, e.g., the only outgoing communication transition in the TCP protocol from SYN_SENT sends ACK and goes to SYN_RECEIVED. The correct logic is to receive SYN first, before sending the ACK and transitioning. The TCP RFC does not textually mention the expected SYN. We only know to expect it because it is illustrated in Figure 6 of the RFC. LLMs can catch these transitions because they can read diagrams. We hypothesize that this ability is a result of fine grained tokenization for training, and also an emergent behavior of LLMs [12, 26]

#### 3.2 Quantitative Experimental Results

Table 1 shows the results for the FSM extraction task using the greedy optimized prompt and APE [30] engineered prompt (TP (True Positives): extracted transitions verified to be correct. FP (False Positives): extracted transitions verified to be incorrect/hallucinated). In the case of DCCP RFC (greedy), we have reported two false positive states (GPT3.5-turbo reports the congestion control identifiers CCID 2 and CCID 3 as system states). We note that while these are not the correct FSM states, during the congestion control negotiation phase, the ‘change L/change R’ events can induce the change of CCID procedure at either the server of the client side, and hence can be construed as a state change.
Figure 2: (best viewed zoomed in) UP: Key artifacts and FSMs extracted using Artifact Miner and PROSPER from (left) RFC4340 - DCCP, (middle) RFC793 - TCP and (right) RFC2637 - PPTP.

Figure 3: Seed prompt and greedily engineered prompt using the train RFC set. The differences are highlighted in orange. The green ‘chunk’ is the chunk of RFC fed to the LLM in a zero shot manner.

Table 1: FSM elements extracted using the greedy prompt and forward generated APE prompt [30] for 4 representative RFCs

<table>
<thead>
<tr>
<th>RFC</th>
<th>Greedy States</th>
<th>Greedy Transitions</th>
<th>APE States</th>
<th>APE Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCCP</td>
<td>12</td>
<td>2</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>TCP</td>
<td>11</td>
<td>0</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>BGP</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>PPTP</td>
<td>6</td>
<td>7</td>
<td>16</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2: Artifacts extracted using Artifact Miner. Example artifacts are in Figure 3.

<table>
<thead>
<tr>
<th>RFC</th>
<th>Full</th>
<th>Partial</th>
<th>FN</th>
<th>FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQTT</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NetBIOS</td>
<td>21</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>DNS</td>
<td>21</td>
<td>0</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>PPP</td>
<td>23</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>PPTP</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3 shows the results for the artifact extraction task using Artifact Miner. We divide true positives into fully extracted, partially extracted and completely missed artifacts. For partial extraction, if Artifact Miner misses more than 10% of total artifact content, we consider it missed. Artifact Miner fails to extract 12 artifacts present in the DNS RFC. Upon close inspection we find that out of these 12, 3 are structurally identical. The 10 unique missed artifacts have structures that use unique symbols that we don’t use in algorithm 1 and including those should increase coverage.

Table 4 compares PROSPER with the results reported in Table 3 of RFCNLP [17]. We have combined the ‘correct’ and ‘partially correct’ transitions reported in RFCNLP under true positives. While we report more true positives, our approach of using foundational models is much better at reducing the number of wrong transitions extracted (false positives). We note the following benefits of using an artifact extractor coupled with large language model for RFC understanding.

- **Generalizability:** Our approach does not rely on a pre-defined state machine grammar, so we can generalize to a diverse set of RFCs.
Table 3: Communication Transitions extracted from the DCCP and TCP protocol using RFCNLP vs PROSPER. For TCP, our approach also extracts data flow transitions which we don’t count but signify with (*)

<table>
<thead>
<tr>
<th>RFC</th>
<th>NeuralICRF+R</th>
<th>Artifact Miner+GPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>DCCP</td>
<td>17</td>
<td>13</td>
</tr>
<tr>
<td>TCP</td>
<td>11</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 3 demonstrates the communication transitions extracted from the DCCP and TCP protocol using different methodologies. For TCP, our approach also extracts data flow transitions which are not counted but marked with (*).

- **Flexibility**: We employ no heuristics at the text understanding stage, and hence it is a more flexible framework for RFC understanding.
- **Applicability**: Using LLMs paves the way for numerous downstream tasks like automated code de-bloating since LLMs process their own outputs.
- **Coverage Enrichment**: We use valuable information available in the RFCs as textual artifacts which increases true positives and reduces false positives.

The need for artifact understanding – A case study of the PPTP RFC2637. RFC documents are often partially complete (partial information in text, rest represented in artifacts), or incomplete (salient protocol information is not described in the RFC). We discuss one such discrepancy in RFC793 in the previous subsection. Similarly, the first half of RFC 2637 (PPTP) describes messages and packet headers, with little mention of FSM states and transitions. The second half describes partial FSMs accurately through multiple artifacts (data flow/state diagrams) but not all the transitions are mentioned in the text. As an example, Figure 18 in the RFC depicts the transitions from ‘established’ to ‘idle’ but the RFC does not mention said transition in text. Figure 20 depicts transitions to and from the ‘wait_cs_ans’ state, but these aren’t specifically mentioned in the text either. These discrepancies translate into the elevated number of false positives observed in Table 1 for the PPTP RFC. The model does a good job of extracting and classifying transitions from all four operational modalities of the protocol (PAC and PNS incoming and outgoing), it fails to extract the ‘wait_cs_ans’ state. We fix this problem by employing artifact based extraction using Artifact Miner.

Artifact Miner extracts all the relevant partial FSM-related artifacts, which we concatenate and feed to GPT3.5-turbo with the engineered greedy prompt described in Figure 2. All 8 states and 25 transitions are extracted with no false positives. We also observe that GPT3.5-turbo is much better at adhering to the JSON return format request made in the prompt. We then subsequently prompted the model to write a python script to draw the protocol FSM. Figure 2 shows some key artifacts and FSMs extracted from the DCCP (RFC 4340), TCP (RFC 793) and PPTP (RFC 2637) RFCs.

4 Limitations and Discussion

Foundational NLP models have shown great potential in natural language understanding, however, they suffer from multiple issues like hallucinations (manifested as false positives in extraction), bias (the model sometimes reports states from a different RFC than the text it was provided), contextual misunderstanding (the model confuses message structures with FSM states in case of PPTP extraction). Here we raise some additional issues that we faced in the context of PROSPER.

- **Return Format Determinism**: Since LLMs are probabilistic models, even the correct output could have multiple formats. Querying the model for transitions in the form of a python data structure could take many forms (list, dict, dict of dicts, JSON etc.). We avoid this by feeding the outputs of the loop to itself in an autoregressive fashion [28] to generate ‘pyGraphviz’ code.

- **End-to-End Extraction**: Unlike [17], PROSPER trades-off the ability to extract information end-to-end in favor of generalizability. We are working to better formalize our approach to remove the human-in-the-loop for general protocol understanding. One way to achieve this is using recently published LLMs [3] that can accept inputs of large context lengths, which would render chunking unnecessary. GPT3.5-turbo has a context length of 2048 tokens. The average number of tokens for an RFC from our representative set is approximately 48K.

- **Addressing lack of tailored benchmark datasets**: LLMs are commonly benchmarked using standardized datasets [19]. These datasets are used both for evaluation and prompt engineering. No such dataset exists for the specialized domain of RFCs, and hence evaluating outputs in terms of accuracy and coverage falls to humans. We prepared a small APE-like [30] dataset for forward generation of prompts, but it can be further improved. We plan to address this in future work.

5 Conclusion and Future Work

We presented PROSPER, a framework to automatically understand and extract protocol specifications from RFCs using LLMs. We select representative RFCs using TF-IDF, union-find, and BERTopic algorithms along with inputs from domain experts, which we utilize for prompt engineering and evaluation. To maximally utilize the information present in RFCs, we develop Artifact Miner, a simple but efficient tool to extract non natural language textual diagrams. We use the engineered prompt, along with text and extracted artifacts to extract states, transitions and events from RFCs. PROSPER achieves 1.3x more true positives and 6.5x less false positives than existing approaches in the FSM extraction task on DCCP RFC. We are working to extend PROSPER in two ways: (i) making it end to end and removing the human in the loop; (ii) incorporating the extracted outputs into multiple downstream tasks like software debloating, intrusion detection, and protocol interface generation.

6 Acknowledgements

This work was funded by the Office of Naval Research (ONR) grant number N00014-18-1-2660. Any expressed opinions, findings and conclusions or recommendations do not necessarily reflect the views of the Office of Naval Research.
References


