# FastRAG: Retrieval Augmented Generation for Semi-structured Data

Amar Abane
*NIST*
Gaithersburg, USA
amar.abane@nist.gov

Anis Bekri
*NIST*
Gaithersburg, USA
anis.bekri@nist.gov

Abdella Battou
*NIST*
Gaithersburg, USA
abdella.battou@nist.gov

***Abstract*—Efficiently processing and interpreting network data is critical for the operation of increasingly complex networks. Recent advances in Large Language Models (LLM) and Retrieval-Augmented Generation (RAG) techniques have improved data processing in network management. However, existing RAG methods like VectorRAG and GraphRAG struggle with the complexity and implicit nature of semi-structured technical data, leading to inefficiencies in time, cost, and retrieval.**

**This paper introduces FastRAG, a novel RAG approach designed for semi-structured data. FastRAG employs schema learning and script learning to extract and structure data without needing to submit entire data sources to an LLM. It integrates text search with knowledge graph (KG) querying to improve accuracy in retrieving context-rich information. Evaluation results demonstrate that FastRAG provides accurate question answering, while improving up to $90\%$ in time and $85\%$ in cost compared to GraphRAG.**

***Index Terms*—Network Management, Large Language Models, Retrieval-Augmented Generation, Generative AI, Prompt Engineering**

## I. Introduction

Efficiently processing and understanding semi-structured data is crucial to improve network management tasks. As networks grow, the diversity of platforms and the sheer volume of data introduce significant challenges. Traditional tools for processing network data [1]–[3] offer some level of utility but fall short in comprehensively extracting and utilizing the full range of information embedded in semi-structured data formats such as logs and configurations. Moreover, the task of correlating these data from different vendors is further complicated by disaggregated network service implementation. Although natural language processing (NLP) and Machine Learning (ML) algorithms have been recently developed to process these data [4], [5], these solutions often lack the flexibility needed to handle diverse data types and formats across varying network environments.

The emergence of Large Language Models (LLMs) has introduced new possibilities for processing and understanding natural language, showcasing their potential in improving network management tools. However, using pre-trained models for generating domain-specific and coherent responses remains challenging. Techniques like Retrieval-Augmented Generation (RAG) [6] have been introduced to enhance LLMs by integrating retrieval methods, with VectorRAG providing context retrieval from textual documents based on semantic similarity [7]. While effective in many scenarios, these traditional RAG systems have limitations, such as loss of critical contextual information due to uniform chunking of documents [8]. GraphRAG [9], a more advanced approach, addresses some of these limitations by leveraging knowledge graphs (KG) to organize information extracted from source documents. Despite its advantages, GraphRAG struggles with queries for accurate information or lacking explicit entities. To overcome these challenges, HybridRAG [8] combines the strengths of both VectorRAG and GraphRAG, offering improved accuracy in retrieving relevant information for LLMs.

Despite these advancements, existing RAG techniques have limitations when applied to network data. The reliance on embedding vectors for context retrieval does not perform well with semi-structured technical data, where implicit information may be hidden behind specific keywords. Moreover, current RAG systems rely on LLMs to extract structured information from source documents by processing them chunk-by-chunk, resulting in increased processing time and cost.

To address these issues, this paper explores a novel RAG approach designed for a cost-effective processing of the large volumes of semi-structured data generated by modern networks. The proposed system (FastRAG) processes source data without requiring the submission of all chunks through an LLM. The contributions of this paper include: the development of prompting methods that use schema learning and script learning to process and structure source data, an algorithm to select sample chunks for these learning prompts, and a retrieval approach that integrates text search and graph querying via Graph Query Language (GQL) for more accurate information retrieval in question answering tasks.

The rest of this paper is organized as follows: Section II reviews related work and discusses the motivation behind developing FastRAG. Section III presents the detailed design of the FastRAG system, including its key components and techniques. The evaluation of FastRAG is discussed in Section IV. Finally, Section V outlines the system's limitations and concludes the paper.

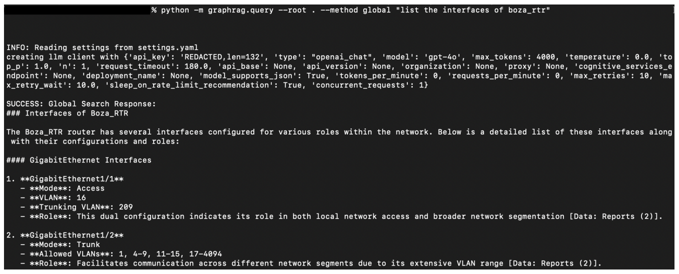
## II. Related Work and Motivation

The integration of LLMs into network management has opened new possibilities, particularly in automating network configuration tasks. Recent studies have explored LLMs' potential to generate network configurations [10]–[12], detect anomalies [13], and convert textural descriptions into formal specifications [14], [15]. These contributions represent the most frequent applications of LLMs to network management, aiming to reduce manual labor and risks of errors. The work proposed in this paper is complementary to these approaches as they may rely on RAG systems to retrieve network-specific information.

The application of RAG within network management has been less explored. RAGLog [16] introduces a novel method for detecting log anomalies using a combination of vector databases and LLMs. This approach allows for the anomalies based on raw log data, adapting to various log sources without extensive preprocessing. While promising for its adaptability, RAGLog encounters challenges related to resource consumption and latency, which could limit its scalability in larger network data. Telco-RAG [17] targets the processing of 3GPP documents. A pipeline with query enhancement demonstrates improvements in the accuracy of LLMs when handling complex questions related to the telecommunication domain.

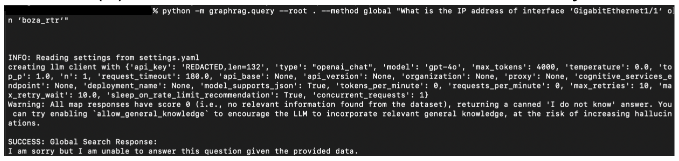
More generally, VectorRAG, the predominant approach in RAG systems, computes vector embeddings for fixed-size chunks and uses semantic similarity to retrieve information relevant to the input query. However, this method struggles with the nature of network data where domain-specific keywords have different meaning in their context. GraphRAG [9] is a more advanced approach that integrates KGs with text chunking and embedding. The LLM is used to extract entities and relationships and summarize communities detected in the KG, providing a more structured retrieval (also uses semantic similarity). Although GraphRAG is efficient in summarizing and reporting tasks (see Figure 1a), it falls short in delivering useful answers, particularly when the query includes specific values, such as names or types, that need to be matched exactly in the retrieval as reported in Figure 1b. HybridRAG [8] combines GraphRAG and VectorRAG. Although this hybrid approach enhances the accuracy and contextual relevance of responses, it inherits the limitations of its base approaches. Another approach uses entity-relationship extraction to create a KG capturing the important information of the source data. The KG is then searched like a database by translating the textual query into a database query [18]. While this allows for accurate retrieval, it can be impractical given the imperfect parsing obtained with LLMs; as shown in Figure 2 where we tested the accuracy of LLM-based configuration parsing against Batfish [2].

To address these limitations, FastRAG extracts entities and their properties from source data while mapping each entity to specific lines of the original text. This method enables accurate retrieval using a KG while allowing for text search to handle vague queries and compensate imperfect entity extraction. Text search, unlike semantic similarity, matches text based on exact wording or structure [19]. Both KG and text searches are performed using GQL queries on a single KG implemented in a graph database. Unlike other methods, FastRAG avoids costly processing of all source data through LLMs by generating JSON schemas and Python code for data structuring and parsing. While demonstrated on logs and configuration data, this approach can be applied to other semi-structured data, such as playbooks and alarms. To the best of our knowledge, this is the first RAG system to rely solely on code generation for data processing.



```
[redacted]% python -m graphrag.query --root . --method global "list the interfaces of boza_rtr"



INFO: Reading settings from settings.yaml
creating llm client with {'api_key': 'REDACTED,len=132', 'type': "openai_chat", 'model': 'gpt-4o', 'max_tokens': 4000, 'temperature': 0.0, 'to
p_p': 1.0, 'n': 1, 'request_timeout': 180.0, 'api_base': None, 'api_version': None, 'organization': None, 'proxy': None, 'cognitive_services_e
ndpoint': None, 'deployment_name': None, 'model_supports_json': True, 'tokens_per_minute': 0, 'requests_per_minute': 0, 'max_retries': 10, 'ma
x_retry_wait': 10.0, 'sleep_on_rate_limit_recommendation': True, 'concurrent_requests': 1}

SUCCESS: Global Search Response:
### Interfaces of Boza_RTR

The Boza_RTR router has several interfaces configured for various roles within the network. Below is a detailed list of these interfaces along
 with their configurations and roles:

#### GigabitEthernet Interfaces

1. **GigabitEthernet1/1**
   - **Mode**: Access
   - **VLAN**: 16
   - **Trunking VLAN**: 209
   - **Role**: This dual configuration indicates its role in both local network access and broader network segmentation [Data: Reports (2)].

2. **GigabitEthernet1/2**
   - **Mode**: Trunk
   - **Allowed VLANs**: 1, 4-9, 11-15, 17-4094
   - **Role**: Facilitates communication across different network segments due to its extensive VLAN range [Data: Reports (2)].
```

(a) Good answer for information summary



```
[redacted]% python -m graphrag.query --root . --method global "What is the IP address of interface 'GigabitEthernet1/1' o
n 'boza_rtr'"



INFO: Reading settings from settings.yaml
creating llm client with {'api_key': 'REDACTED,len=132', 'type': "openai_chat", 'model': 'gpt-4o', 'max_tokens': 4000, 'temperature': 0.0, 'to
p_p': 1.0, 'n': 1, 'request_timeout': 180.0, 'api_base': None, 'api_version': None, 'organization': None, 'proxy': None, 'cognitive_services_e
ndpoint': None, 'deployment_name': None, 'model_supports_json': True, 'tokens_per_minute': 0, 'requests_per_minute': 0, 'max_retries': 10, 'ma
x_retry_wait': 10.0, 'sleep_on_rate_limit_recommendation': True, 'concurrent_requests': 1}
Warning: All map responses have score 0 (i.e., no relevant information found from the dataset), returning a canned 'I do not know' answer. You
 can try enabling `allow_general_knowledge` to encourage the LLM to incorporate relevant general knowledge, at the risk of increasing hallucin
ations.

SUCCESS: Global Search Response:
I am sorry but I am unable to answer this question given the provided data.
```

(b) Wrong answer for accurate question

Fig. 1: Example questions and answer with GraphRAG

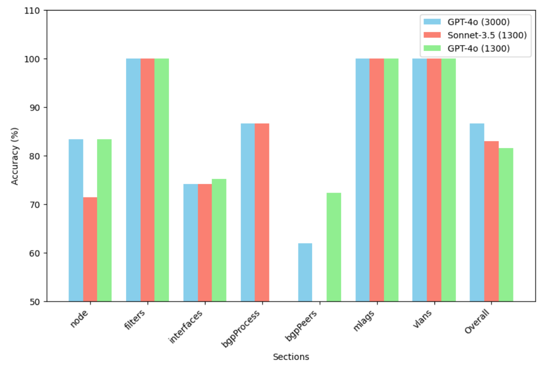


Fig. 2: Accuracy of an LLM in parsing configuration files

## III. FastRAG Design

We designed a hybrid approach for both information extraction and retrieval. Instead of using indiscriminate chunking or user-defined schema, our method relies on converting data into a simple automatically-generated JSON structure, while maintaining a link to the original data in a KG [20]. The RAG further relies on LLMs to interact with the KG, utilizing GQL query generation to fetch and interpret information.

Given the extensive and continually updating nature of network data, minimizing processing time and cost is essential. To achieve this, we leverage prompt engineering findings which indicate that LLMs are more effective at generating code rather than directly extracting information into structured formats. Consequently, we introduce schema learning and script learning techniques, where we sample the source data and iteratively task the LLM to generate JSON schemas to structure the information and Python code to extract entities.

An overview of the FastRAG architecture is illustrated in Figure 3 and its components are discussed below.

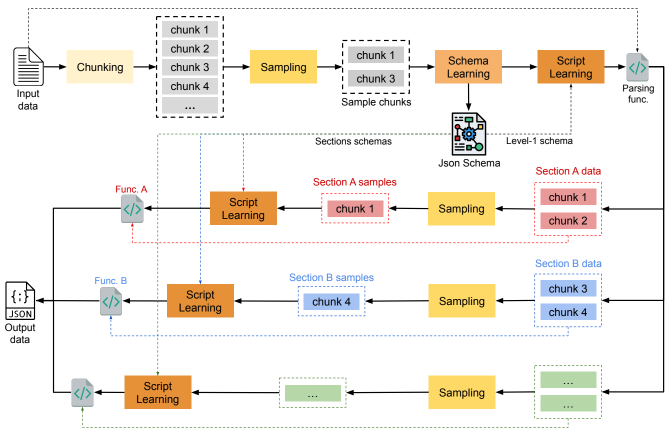


Fig. 3: FastRAG architecture

## A. *Chunk sampling*

Even with expansive context windows of 100k to 200k tokens, LLMs may struggle in processing large data within a single prompt due to context retention limitations [21]. As a result, it is common practice to divide the data into small chunks, typically of uniform size with an overlap between consecutive chunks, and submit these chunks to the LLM through multiple calls. However, submitting all the chunks can be inefficient. To address this, we exploit the repetitive formatting present in semi-structured data, selecting a representative set of chunks—or sample chunks—that effectively covers the syntax of the source data [13].

The chunk sampling process involves two key procedures: keyword extraction and chunk selection.

*1) Keyword extraction:* We draw inspiration from NLP techniques to identify the most meaningful terms in the source data [4]. The process begins with the preprocessing of the text, where punctuation is removed and the content is tokenized into individual words. Given that English is the predominant language used in most semi-structured network data, it is the language adopted for this process. After preprocessing, the text is split into lines, and a matrix representation is created based on word frequency, where each row corresponds to a line and each column to a unique term from the entire text. The matrix, which reflects the frequency of each term in each line, serves as the input for the K-means clustering algorithm. This algorithm groups the lines into $n_c$ number of clusters based on the similarity of their term frequency patterns. Within each cluster, the $n_t$ closest terms to the centroids are selected as keywords. The output is a set of keywords that encapsulate the primary terms present in the text corpus.

*2) Samples selection:* We designed an algorithm that selects the smallest set of chunks that collectively contain the complete set of extracted keywords. However, merely adding chunks until all keywords are covered does not guarantee the selection of the (near) minimum number of sample chunks. To address this, the algorithm selects chunks based on their entropy that provides an estimate of the diversity introduced by each chunk.

The procedure (see Algorithm 1) begins with the preprocessing of the chunks (as in the previous step) where only the relevant keywords are retained. The preprocessed chunks are then used to compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors, representing the importance of terms within each chunk. Next, the Shannon entropy for each chunk is calculated to gauge the informational diversity it contributes. The algorithm iteratively selects a subset of chunks to maximize term coverage, focusing on those that introduce new terms and the most diversity, until the desired coverage threshold—fixed at *1* in all cases—is reached.

**Algorithm 1** Samples Selection

```
Input: $chunks, threshold, keywords$
$tokenized \leftarrow$ tokenize/filter chunks w/ $keywords$
$tfidf \leftarrow$ TF-IDF vectors of $tokenized$
$entropies \leftarrow$ entropy for each chunk in $tokenized$
$n\_chunks, n\_terms \leftarrow$ from $tfidf$
$selected, covered \leftarrow \emptyset, \emptyset$
while $\frac{|covered|}{n\_terms} < threshold$ do
    $gains \leftarrow \emptyset$
    for each $i \in [1, n\_chunks]$ do
        if $i \notin selected$ then
            $nw\_trms \leftarrow$ terms in $tokenized[i]$ not in $covered$
            $gain \leftarrow |nw\_trms| \times entropies[i]$
            $gains \leftarrow gains \cup (gain, i)$
        end if
    end for
    if $gains \neq \emptyset$ then
        $best \leftarrow$ chunk w/ max gain from $gains$
        $selected \leftarrow selected \cup best$
        $covered \leftarrow covered \cup$ terms in $tokenized[best]$
    else
        break
    end if
end while
$output \leftarrow$ chunks corresponding to $selected$
Output: $output$
```

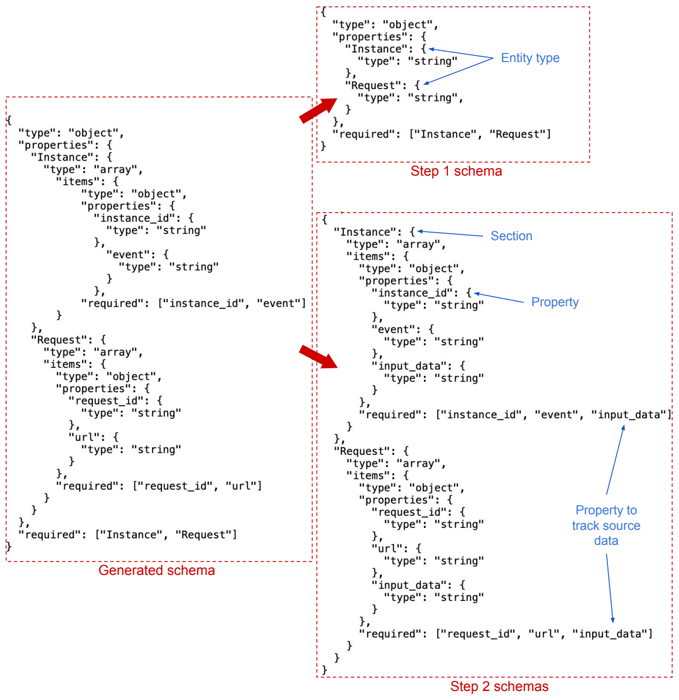
The number of sample chunks selected is directly influenced by the number of extracted keywords, which can be adjusted by varying the parameters $n_c$ (number of clusters) and $n_t$ (number of terms). This process is discussed in greater detail in Section IV.

## B. *Schema learning*

Building on the selected sample chunks, we develop a prompt strategy that guides the LLM in identifying entity types and their properties, focusing on schema extraction rather than specific entities. The process (Figure 5) begins with the first sample chunk, where the LLM is prompted to identify and structure the entity types and properties into a JSON schema. In subsequent prompts, the previous JSON schema is iteratively refined by submitting new chunks to the LLM. To avoid complexity, the LLM is instructed to limit high levels of nesting and refrain from creating objects with an excessive number of properties.

After each prompt, the strategy includes a verification step to ensure that the JSON schema generated by the LLM is well-formed. If the schema contains errors, the LLM is asked to correct it based on the provided error message. Each prompt call can be retried up to $N = 4$ times.

From the finalized schema, two types of objects are extracted, as illustrated in Figure 4. The first is the *Step 1* schema, which is a JSON schema that includes only the level-1 entity types—referred to as sections. Each entity type includes a description[1] and a string property that contains the source data lines defining the objects belonging to that section. The second, *Step 2* schema, is a map where each section is associated with an array where objects correspond to the full schema of that section. This division into Step 1 and Step 2 schemas allows for a more structured approach to data extraction, as discussed below.



```
{
  "type": "object",
  "properties": {
    "Instance": {
      "type": "array",
        "items": {
            "type": "object",
            "properties": {
              "instance_id": {
                "type": "string"
              },
                "event": {
                  "type": "string"
                }
              },
            "required": ["instance_id", "event"]
        }
    },
    "Request": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "request_id": {
            "type": "string"
          },
          "url": {
            "type": "string"
          }
        },
        "required": ["request_id", "url"]
      }
    }
  },
  "required": ["Instance", "Request"]
}
```

```
{
  "type": "object",
  "properties": {
    "Instance": {
      "type": "string"
    },
    "Request": {
      "type": "string",
    }
  },
  "required": ["Instance", "Request"]
}
```

```
{
  "Instance": {
    "type": "array",
    "items": {
      "type": "object",
      "properties": {
        "instance_id": {
          "type": "string"
        },
        "event": {
          "type": "string"
        },
        "input_data": {
          "type": "string"
        }
      },
      "required": ["instance_id", "event", "input_data"]
    }
  },
  "Request": {
    "type": "array",
    "items": {
      "type": "object",
      "properties": {
        "request_id": {
          "type": "string"
        },
        "url": {
          "type": "string"
        },
        "input_data": {
          "type": "string"
        }
      },
      "required": ["request_id", "url", "input_data"]
    }
  }
}
```

Fig. 4: Example of processing of generated schema

## C. Script learning

Similar to the approach used in schema learning, the script learning prompt strategy (Figure 5) begins by instructing the LLM to generate parsing functions based on the first sample chunk. The LLM is then prompted iteratively to refine the previous function code by submitting new sample chunks. To ensure both the syntactic and functional correctness of the generated code, the prompt includes a verification step where the code is executed with the sample chunk used to generate it. If the code contains errors, the LLM is asked to correct them based on the feedback message. Similarly, each prompt call can be retired up to N times.

In the Step 1, the prompt instructs the LLM to map each line in the source data to its corresponding section according to the schema. Once the function is generated, it is used to parse the entire source data and divide it into sections. After that, each section is further split into fixed-size chunks, and the sampling process is repeated for each section. At this stage, each section has its own schema (contained in the Step 2 schema object) and a set of sample chunks, enabling it to be processed independently, following the same method as in Step 1.

In the Step 2, script learning is utilized to generate a specific parsing function for each section. Once these functions are generated, the data within each section is processed using the corresponding parsing function. Processing sections independently enables the LLM to concentrate more effectively on each entity type and also allows for a more targeted refinement of the sample chunks by re-executing the sampling process for each section.

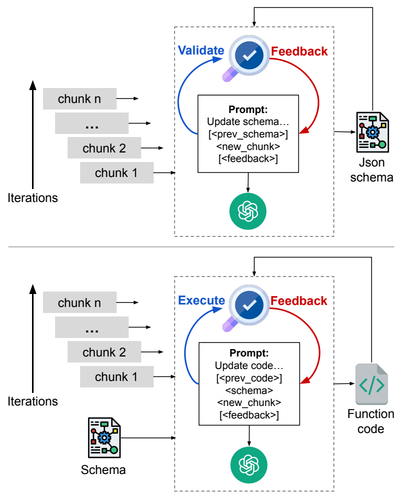


Fig. 5: Schema learning (top) and script learning (bottom)

## D. Information Retrieval

The extraction process results in a JSON object where each entity is represented, and within this object, an *input_data* property contains the lines from the source data that define the entity. This JSON structure serves as the foundation for the KG creation and the information retrieval.

*1) KG Creation:* Each entity identified in the JSON object is inserted as a node within the KG, with the entity's type serving as the node's label. Simple-type properties of the entity are directly assigned as properties of the node, while properties

[1]Not shown in the Figure

that are themselves objects are inserted as child nodes, connected to their parent node. Additionally, for each line within the *input_data* property of each entity, a corresponding node is created and linked to the parent entity node. These input data nodes are used for text searches leveraging NLP methods [22].

Several retrieval strategies have been defined and tested to interact with this KG, as discussed below.

*2) KG Querying (Graph):* In this strategy, a prompt is used to provide the LLM with the schema of the KG. The LLM is then instructed to generate a syntactically correct GQL statement that can answer the input textual query. Once generated, this GQL statement is executed and the results are further interpreted by the LLM to generate the answer.

*3) Text Search (Text):* This strategy employs a slightly different prompting approach. The LLM is given examples of text search features (e.g., from the documentation), typically using regular expressions, and is then instructed to generate a GQL statement that exclusively utilizes text search features to address the input textual query. After the statement is executed the results are interpreted by the LLM.

*4) Combined Querying (Combined):* This strategy involves executing both the KG querying and text search prompts in parallel for a given input query. The results from each method are directly returned without LLM interpretation. These results are then provided to the LLM to synthesize the final answer based on the combined context.

*5) Hybrid Querying (Hybrid):* This strategy merges the capabilities of both KG querying and text search. The prompt provided to the LLM includes both the KG schema and text search examples, instructing the LLM to generate a GQL statement that can leverage any relevant features from both methods to answer the input query.

## IV. Evaluation

We designed an experimental setup to evaluate the performance of FastRAG. Our implementation utilized the OpenAI GPT-4o model as the LLM, Neo4j as the graph database to store and query the KG using Cypher GQL, and the Langchain library for LLM prompting.

The evaluation was conducted using two different data sources that represent typical network data: logs and configurations. For the logs, we used a single file containing 2000 lines of OpenStack logs [23], resulting in 1307 chunks of $\sim$ 1000 tokens each. For the configurations, we used 13 files of Cisco device configurations, totaling 2100 lines and divided into 19 chunks of $\sim$ 1000 tokens each. These datasets were selected to create a manageable use case that allowed for manual verification of the results.

### A. Information Extraction

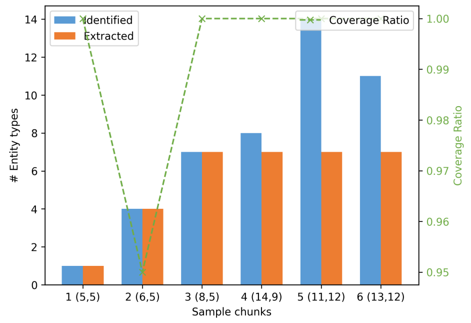
*1) Step1 schema and script learning:* When using FastRAG, determining the optimal number of sample chunks for each dataset is crucial, as it directly impacts the quality of schema and script learning. If too few samples are used, the system may overlook important entity types, leading to incomplete parsing. On the other hand, using too many samples can result in overly complex schemas with unnecessary entity types. Additionally, we observed that the optimal sample size for Step 1 does not necessarily translate to Step 2, as the keywords extracted in Step 1 might not fully align with the relevant keywords for each section in Step 2. Consequently, finding the best number of sample chunks for both steps independently is essential.

Samples size is a hyperparameter in our design, requiring experimentation with different sizes by varying $n_c$ and $n_t$ to identify the sample size that yields the best information extraction. To streamline this process we defined a *coverage* metric. This metric measures the ratio of lines in the source documents that are included in the *input_data* of all extracted objects.

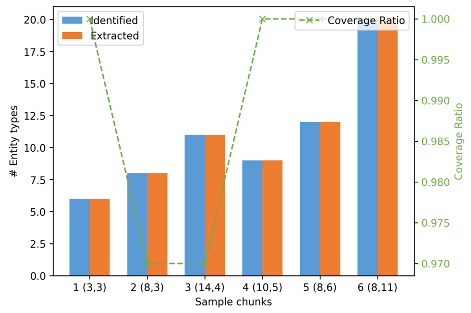
To determine the optimal number of chunks for Step 1, we varied the sample size from 1 to M by adjusting $n_c$ and $n_t$ to achieve the desired number of chunks. We then ran the schema extraction process, generated the corresponding parsing function, and calculated the coverage ratio by executing the function on the source data. Figure 6 reports the results of this procedure. The coverage ratio reaches 1 with just one sample chunk; however, this is because the schema produced is overly simplistic, identifying only one entity type. Therefore, it is also important to consider the completeness and relevance of the extracted schema by examining the number of identified entity types. The goal is to select the most comprehensive schema that also leads to (near) 100% coverage ratio. Another observation is that the LLM tends to add more entity types when more chunks are used, possibly due to the instruction to avoid too many properties in a single entity type. However, we found that overlapping or superfluous entity types are not a significant concern. During script learning, the LLM often fails to find efficient ways to parse these extra types, resulting in empty sections that are ignored in Step 2. This occurs because the LLM prioritizes achieving a 100% coverage ratio, sometimes at the expense of extracting all entity types. To highlight this, Figure 6 differentiates between the number of entity types identified and the number of entity types extracted with the generated parsing function. Based on these observations, we determined the optimal number of sample chunks for each dataset: 4 chunks out of 1307 for the logs dataset and 4 chunks out of 19 for the configuration dataset.

In addition to aiding in the selection of the best sample size, the coverage ratio also serves as an empirical metric to evaluate the quality of extraction of our RAG system.

To assess the efficiency of using the LLM in our method, we measured the volume of input and output characters exchanged with the LLM in both schema and script learning prompts, shown in Figure 7. The input character count for both prompts is similar, particularly in the first half of the chart. However, output characters, which weight more in the model's pricing and are limited to 4000 tokens per output, show different trends. Script learning tends to use fewer output characters, while schema learning's output increases significantly with

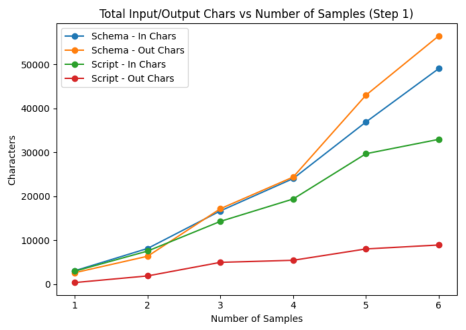


(a) Logs

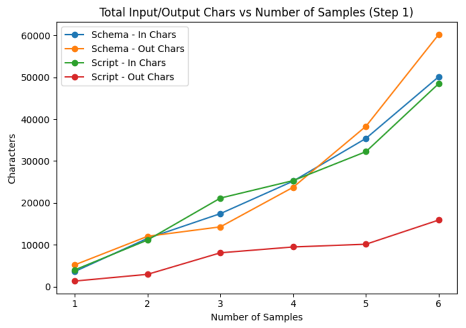


(b) Configurations

Fig. 6: Effect of samples size on extraction (Step 1)

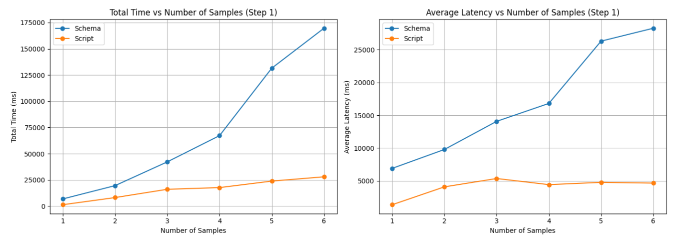


(a) Logs

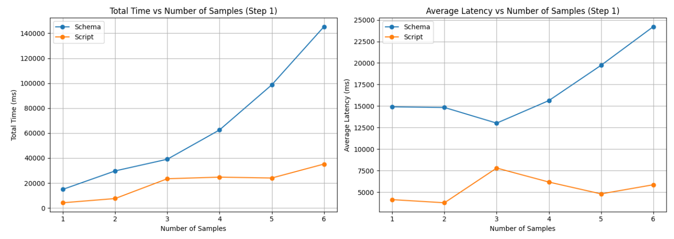


(b) Configurations

Fig. 7: Effect of samples size on input and output size (Step 1)



(a) Logs



(b) Configurations

Fig. 8: Effect of samples size on prompt execution time (Step 1)

larger sample sizes. Nonetheless, with the sample size fixed to 4 for both datasets, the output character count for schema learning remains within a reasonable range, indicating that the selected sample size is a cost-effective choice.

We also measured the average latency and total time of the prompts, depicted in Figure 8. These results indicate that the FastRAG processes the entire source data quickly compared to submitting all the data through the LLM. Notably, no prompt call required a retry for any sample size. Schema extraction typically takes longer than code generation, as schemas can be more complex and detailed. However, this process only occurs once at the beginning and remains unchanged unless significant changes, such as new services or applications, are introduced. Similar to character counts, the highest latency and total time values were associated with larger sample sizes, which are beyond the samples size we selected. Only a few requests required retries, mainly in cases where the schema was invalid due to errors like incorrect property types or missing required properties. An unusual spike in the average latency during script learning for the configuration dataset likely resulted from the LLM quickly producing an overly complicated schema that required more time or retries during script learning. This likely led to an imperfect parsing function, as indicated by the coverage ratio of 97%.

In general, when the average latency does not increase with additional sample chunks, it suggests that the added sample chunk did not introduce significant new information, allowing the LLM to quickly return an updated schema without much additional processing time.

*2) Step 2 script learning:* The process of determining the number of sample chunks for Step 2 script learning is similar to that of Step 1, but it is somewhat simpler. In Step 2, a parsing function is generated for each section, and no schema extraction is involved. As a result, the focus is solely on identifying the number of chunks needed to achieve a coverage ratio close to 100%. For simplicity, we fixed the same number of sample chunks for all sections, even though the optimal number might vary between sections. However, the results demonstrate that the number of sample chunks does not need to be precisely accurate; achieving near 100% coverage is possible even with one or two additional chunks beyond what is strictly necessary.

Table I presents the performance metrics and parameters in Step 2 for both datasets. It is worth noting that the prompts generally required more retries than in Step 1. This increase in retries is due to the need to extract detailed properties for each section, which demands a higher level of precision and accuracy from the LLM.

## B. Question Answering

We evaluate the retrieval effectiveness of FastRAG through question answering. We start by randomly selecting evaluation chunks in Step 1 that were different from those used as sample chunks. We then employed two different LLMs, GPT-4o-mini and Claude-3.5-Sonnet, to generate questions and typical answers based on these evaluation chunks. The questions were submitted to FastRAG and we manually checked the accuracy of the answers. This manual verification is feasible given the relatively small size of the datasets and the manageable number of questions, especially after gaining familiarity with the datasets.

We categorized each answer with one of the following labels: (-) for incorrect answers, (+) for correct answers that do not provide additional information, and (++) for correct answers that include detailed information.

We evaluated FastRAG with the four proposed retrieval methods—Graph, Text, Combined, and Hybrid—on the two studied datasets, logs (Table II) and configurations (Table III).

For the logs dataset, Graph querying retrieval struggled with questions requiring a broader contextual understanding, such as identifying types of logs or understanding the function of *nova-compute.log* entries. This method often failed to retrieve comprehensive information, especially when abstract or holistic answers were needed. Text retrieval performed better in retrieving specific details directly from the data, such as typical entries in *nova-api.log*. However, it fell short when exhaustive details were required, as seen in queries about common HTTP methods in the API logs. Combined and Hybrid retrieval outperformed both Graph and Text methods by aggregating information from multiple sources, providing context-rich and detailed answers. Combined retrieval excelled, particularly in identifying the most common HTTP methods and providing a complete explanation of log functions. In certain cases, guiding Text retrieval with query reformulations, such as broadening the search criteria, improved accuracy.

In the configurations dataset, Graph retrieval performed better, particularly in identifying devices and extracting specific configuration details like IP addresses and route-map values. This highlights its effectiveness in relational and accurate data contexts. However, it showed limitations in queries about access lists, possibly due to incomplete data extraction. Text retrieval, meanwhile, frequently produced incomplete results, particularly for more complex queries such as identifying IP addresses or autonomous systems. Once again, Combined retrieval outperformed both individual models, providing more complete answers for queries like listing prefix-lists, thanks to its ability to leverage both text and graph-based methods. Hybrid retrieval, though effective in most cases, occasionally struggled with certain queries, such as access-list matches, pointing to a sensitivity in query formulation.

Overall, combining both graph-based and text-based retrieval provides the most effective solution for question answering tasks in network data, allowing for both precision and completeness. However, while Hybrid retrieval currently underperforms compared to the Combined approach, it shows promising potential for improvement through techniques such as prompt refinement and the use of few-shot examples.

## C. Cost and Speed Improvement

We conducted an experiment to compare the time and cost of FastRAG with those of the GraphRAG system discussed in Section II, through the processing of logs and configuration data.

To maintain a manageable experiment size while still having enough source data for comparison, we adapted the original datasets as follows: the logs dataset was reduced from 2000 to 500 lines, resulting in 78 chunks (of $\sim$ 1000 tokens) for GraphRAG and 319 chunks (of $\sim$ 1000 tokens) for FastRAG (Step 1). The configuration dataset was replaced with a larger dataset [24], comprising five configuration files, totaling 4,400 lines, which yielded 67 chunks of $\sim$ 1000 tokens for both GraphRAG and FastRAG (Step 1).

To ensure similar conditions for the comparison, we adjusted the parameters of GraphRAG as follows: chunk size was set to 1000 tokens, chunk overlap was set to 50 tokens ($\sim$ 5 lines), and the number of concurrent requests was limited to one.

We measured the total time and cost[2] required to run both RAG systems until the KG creation. For FastRAG, the measured time included the $(n_c, n_t)$ hyper-parameters search for the fixed number of sample chunks for each dataset. The procedure of determining the optimal sample size through trial and coverage was not included in the total time. The best sample chunk sizes were fixed as follows. For the configurations dataset: 6 sample chunks in Step 1 (achieving 98.7% coverage) and 4 sample chunks in Step 2 (achieving 94.6% coverage). For the logs dataset: 4 sample chunks in Step 1 (achieving 100% coverage) and 4 sample chunks in Step 2 (achieving 95% coverage).

Table IV presents the results for both evaluated systems. As expected, the difference in cost and time between FastRAG and GraphRAG is significant, even with these relatively small datasets. FastRAG's improvement is evident, particularly when considering that the generated parsing functions do not need

[2]Cost measured from the API usage provided on the OpenAI dashboard.

TABLE I: Step 2 parameters and performance

| | Samples Size | Entity Types | Total Requests | Total In Chars | Total Out Chars | Total Time (s) | Coverage |
|---|---|---|---|---|---|---|---|
| **Logs** | 4 | 7 | 33 | 153833 | 55763 | 182 | 98% |
| **Configurations** | 2 | 9 | 20 | 56019 | 42945 | 111 | 100% |

TABLE II: Q&A evaluation results for the logs dataset

| Question | Graph | Text | Combined | Hybrid |
|---|---|---|---|---|
| What type of log APIs are present in the data | + | - | + | + |
| What is the primary function of the nova-compute.log entries | - | + | ++ | + |
| What kind of information is typically logged in the nova-api.log entries | - | ++ | + | + |
| What are the most common HTTP methods observed in the api logs | + | - | ++ | + |
| What types of IP addresses are frequently seen in the log entries | + | + | ++ | + |
| What kind of metadata-related requests are present in the logs | - | ++ | ++ | - |
| Are there any indications of VM state changes in the logs | - | + | + | ++ |
| What is the typical response time for API requests in the logs | + | - | + | + |
| Are there any error status codes present in the log entries | - | + | + | - |
| Are there any recurring patterns in the server requests | ++ | - | + | ++ |
| What is the latest status of the image with ID ‘0673dd71...‘ | - | - | - | - |
| What was the response status for the last GET request to ‘/v2/54fadb41.../servers/detail‘ | + | + | + | + |
| How long the latest GET request to ‘/openstack/v2‘ takes | - | + | + | - |
| What events were logged for the instance with ID ‘af9d460c-89bf-...‘ | - | + | + | + |
| How many times was the GET request to ‘/v2/54fadb41.../servers/detail‘ made | + | - | + | + |
| What information can be found in the metadata for the instance logs | + | - | + | + |
| What server IP addresses were involved in requests about metadata | - | - | - | - |
| What was the response length for the GET request to ‘/openstack/2013-10-17/meta_data.json‘ | + | + | + | + |

TABLE III: Q&A evaluation results for the configurations dataset

| Question | Graph | Text | Combined | Hybrid |
|---|---|---|---|---|
| How many devices are in the network | + | - | + | + |
| What are the interfaces on ‘as1border1‘ | + | - | + | + |
| What is the IP address of interface ‘GigabitEthernet0/0‘ on ‘as1border1‘ | + | - | + | - |
| What is the typical local-preference value set in the route maps | + | + | + | + |
| What autonomous systems (AS) are mentioned in the route maps | + | - | + | + |
| What is the most common metric value set in the route maps | + | + | + | + |
| What community value is assigned in the route-map ‘as2_to_as1‘ | + | - | + | + |
| What is the local preference value set in the route-map ‘as2_to_as1‘ | + | - | + | + |
| Which access-list permits IP traffic for the host ‘1.0.2.0‘ | - | + | + | - |
| What prefix-list is matched in the route-map ‘as2_to_as3‘ | + | - | + | - |
| What are the prefix-lists configured | + | + | ++ | + |

to be regenerated as long as the source data contains the same type of information. These functions can also be updated periodically to refine the information extraction process.

Note that GraphRAG uses the LLM to generate entity descriptions and community reports during the information extraction process, which are essential for the question-answering. Additionally, while we limited GraphRAG to one concurrent LLM request for fairness in this comparison, in practical scenarios, the processing time could be reduced by utilizing parallel requests. This, however, does not reduce the cost.

TABLE IV: Comparison of FastRAG and GraphRAG

| Dataset | System | Time (min) | Cost (USD) |
|---|---|---|---|
| Logs | FastRAG | 3.9 | 0.91 |
| | GraphRAG | 40.0 | 6.00 |
| Configurations | FastRAG | 2.0 | 1.10 |
| | GraphRAG | 6.3 | 4.00 |

## V. Conclusion

FastRAG is a cost-effective and time-efficient method for processing semi-structured network data, demonstrated on network logs and configurations. However, its reliance on sample selection introduces a risk of missing small portions of information ($< 5\%$ over the reported results). Additionally, as an LLM-powered system, it introduces variability, requiring continuous verification to ensure accuracy. The current design also lacks entity relationship extraction, which could improve retrieval at a slightly higher cost.

Despite these limitations, FastRAG offers promising performance and resource savings for processing large and frequently changing data, particularly given the rapidly improving LLM capabilities. Interestingly, at the time this study was completed, OpenAI announced a new version of GPT-4o, which reliably adheres to developer-supplied JSON schemas [25], and a new series of models designed to spend more time on reasoning before generating their answers [26].

## Disclaimer

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST, nor does it imply that the products mentioned are necessarily the best available for the purpose.

## References

[1] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1298. [Online]. Available: https://doi.org/10.1145/3133956.3134015

[2] M. Brown, A. Fogel, D. Halperin, V. Heorhiadi, R. Mahajan, and T. Millstein, "Lessons from the evolution of the batfish configuration analysis tool," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 122–135. [Online]. Available: https://doi.org/10.1145/3603269.3604866

[3] Z. Peng, G.-J. Aaron, Z. Yueshang, Y. Huang, L. Xu, and L. Hao, "Differential network analysis," in *19th USENIX Symposium on Networked Systems Design and Implementation*, 2022, pp. 601–615.

[4] Y. Huo, Y. Su, C. Lee, and M. R. Lyu, "Semparser: A semantic parser for log analysis," 2023. [Online]. Available: https://arxiv.org/abs/2112.12636

[5] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "Config2Spec: Mining network specifications from network configurations," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 969–984. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/birkner

[6] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2312.10997

[7] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[8] B. Sarmah, B. Hall, R. Rao, S. Patel, S. Pasquali, and D. Mehta, "Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction," 2024. [Online]. Available: https://arxiv.org/abs/2408.04948

[9] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," 2024. [Online]. Available: https://arxiv.org/abs/2404.16130

[10] E.-D. Jeong, H.-G. Kim, S. Nam, J.-H. Yoo, and J. W.-K. Hong, "S-witch: Switch configuration assistant with llm and prompt engineering," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–7.

[11] R. Mondal, A. Tang, R. Beckett, T. Millstein, and G. Varghese, "What do llms need to synthesize correct router configurations?" in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, ser. HotNets '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 189–195. [Online]. Available: https://doi.org/10.1145/3626111.3628194

[12] C. Wang, M. Scazzariello, A. Farshin, D. Kostic, and M. Chiesa, "Making network configuration human friendly," 2023. [Online]. Available: https://arxiv.org/abs/2309.06342

[13] V.-H. Le and H. Zhang, "Log parsing with prompt-based few-shot learning," 2023. [Online]. Available: https://arxiv.org/abs/2302.07435

[14] D. Donadel, F. Marchiori, L. Pajola, and M. Conti, "Can llms understand computer networks? towards a virtual system administrator," 2024. [Online]. Available: https://arxiv.org/abs/2404.12689

[15] B. Ifland, E. Duani, R. Krief, M. Ohana, A. Zilberman, A. Murillo, O. Manor, O. Lavi, H. Kenji, A. Shabtai, Y. Elovici, and R. Puzis, "Genet: A multimodal llm-based co-pilot for network topology and configuration," 2024. [Online]. Available: https://arxiv.org/abs/2407.08249

[16] J. Pan, S. L. Wong, and Y. Yuan, "Raglog: Log anomaly detection using retrieval augmented generation," 2023. [Online]. Available: https://arxiv.org/abs/2311.05261

[17] A.-L. Bornea, F. Ayed, A. D. Domenico, N. Piovesan, and A. Maatouk, "Telco-rag: Navigating the challenges of retrieval-augmented language models for telecommunications," 2024. [Online]. Available: https://arxiv.org/abs/2404.15939

[18] Tomaž Bratanič, "Knowledge Graphs & LLMs: Multi-Hop Question Answering," Accessed on 09/12/2024. [Online]. Available: https://neo4j.com/developer-blog/knowledge-graphs-llms-multi-hop-question-answering/

[19] Alex Thomas, "Hybrid Retrieval for GraphRAG Applications Using the Neo4j GenAI Python Package," Accessed on 09/12/2024. [Online]. Available: https://neo4j.com/developer-blog/neo4j-genai-python-package-graphrag/

[20] H. Khorashadizadeh, F. Z. Amara, M. Ezzabady, F. Ieng, S. Tiwari, N. Mihindukulasooriya, J. Groppe, S. Sahri, F. Benamara, and S. Groppe, "Research trends for the interplay between large language models and knowledge graphs," 2024. [Online]. Available: https://arxiv.org/abs/2406.08223

[21] OpenAI, "Optimizing LLMs for accuracy," Accessed on 09/12/2024. [Online]. Available: https://platform.openai.com/docs/guides/optimizing-llm-accuracy/llm-optimization-context

[22] Apache, "Apache Lucene," Accessed on 09/12/2024. [Online]. Available: https://lucene.apache.org

[23] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for ai-driven log analytics," 2023. [Online]. Available: https://arxiv.org/abs/2008.06448

[24] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX Association, Apr. 2013, pp. 99–111. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/kazemian

[25] OpenAI, "Introducing Structured Outputs in the API," Accessed on 09/12/2024. [Online]. Available: https://openai.com/index/introducing-structured-outputs-in-the-api/

[26] ——, "Introducing OpenAI o1," Accessed on 09/12/2024. [Online]. Available: https://openai.com/o1/