Supercharging Proactive Network Maintenance by Leveraging Generative AI

A technical paper prepared for presentation at SCTE TechExpo24

Santhana Chari, Ph.D.
VP, Broadband Analytics and Data Science
OpenVault
schari@openvault.com

Mahesh Kanase
Sr. Data Science Engineer
Ksolves
mahesh.kanase@openvault.com

Table of Contents

Title	Page Number
1. Introduction	3
2. Interactive LLM Applications	3
3. LLM Performance Improvement Techniques	4
3.1. Fine-tuning	5
3.2. Retrieval Augmented Generation (RAG)	5
3.3. Comparing Fine-tuning and RAG	6
4. Evaluating RAG Performance	7
4.1. End-to-end Performance Evaluation	7
4.2. RAG Context Efficacy and Relevancy	10
5. Leveraging Dynamic Data	12
6. Conclusion	13
Abbreviations	15
Bibliography & References	15

List of Figures

Title	Page Number
Figure 1 - RAG processing pipeline and architecture.	6
Figure 2 - METEOR and Cosine similarity computed for the 27 test queries	9
Figure 3 - RAG Evaluation system	10
Figure 4 - Sample query, top two retrieved contexts, and the responses generated by the LLM	11
Figure 5 - Generalized Retriever approach to use custom and third-party APIs	13

List of Tables

Title	Page Number
Table 1 - Comparison of RAG and Fine-tuning approaches to LLM optimization	7
Table 2 - RAGAS metrics computed for various Top-K chunks retrieved from the vector datastore	12

1. Introduction

The Cable and Telecommunication industry has been at the forefront of collecting staggering amounts of data given their end-user subscriber base runs into hundreds of millions of users. The data is collected from devices that are deployed in both core and edge of the network, and at consumer residences that are geographically distributed. Large volumes of data thus collected spans various categories ranging from consumer specific data, aggregated network utilization and usage data, and operational data from hardware devices and software micro-services. This data collection has historically used legacy protocols such as simple network management protocol (SNMP) and Internet Protocol Detail Record (IPDR). Most legacy data collection frameworks use *pull-models* where the collectors periodically poll the devices to collect and aggregate the data. But with increased emphasis on network automation and orchestration driven by distributed access architectures, there is a growing impetus to migrate to more modern model-driven telemetry approaches where the endpoints are configured to stream the data using *push-models* that are based on standard specifications in a vendor agnostic fashion.

With the availability of copious amounts of data comes the natural question of effective approaches to leverage the data to optimize network planning and operations. In the past statistical methods and models that were originally invented several decades ago were used to analyze the data to perform *diagnostic* and *predictive* analysis. Diagnostic analysis was mainly used to identify root causes of issues in the network based on historical or real-time data. Predictive analysis, on the other hand, used historical data to estimate future load on the network and prepare the network to meet the quality of experience requirements [ulm-2019]. More recently these statistical approaches were replaced by classical Machine Learning (ML) approaches that use classification and regression techniques using supervised and unsupervised learning techniques [volpe-2021], [righetti-2023].

In this paper, we primarily focus on the use of artificial intelligence (AI) tools, and more specifically Generative AI tools to leverage the vast repository of data and to address proactive network management (PNM). Rapid and recent advances in the transformer models [vaswani-2017] have completely changed the paradigm on how a non-technical user can interact with complex software systems employing large language models (LLM) using only simple natural language queries. We have addressed the problem of how LLMs that have been pre-trained for very general tasks can be customized to analyze and leverage data specific to certain domains, in this case the knowledge base and data specific to the cable industry.

The rest of the paper is organized as follows. Section 3 provides details on techniques to augment LLMs with private, application-specific data; we discuss and compare two different techniques, namely fine-tuning and retrieval augmented generation (RAG). We also present details on how to evaluate the efficacy of RAG applications. We address both end-to-end evaluation of RAG applications using a set of pre-defined prompts and expected responses, as well as present techniques to evaluate the individual building blocks of a RAG system, namely, the embedding, chunk size, number of chunks, etc. Finally, we present details on generalized retrievers using the LangChain framework that can leverage local or third-party data to build advanced retrieval systems.

2. Interactive LLM Applications

The ease of interaction with LLMs using natural language queries has resulted in a proliferation of applications and software tools to rapidly build applications that support querying the information that was embedded in the LLM models during the pre-training phase. Such virtual assistant applications can be used to improve the efficiency of customer support agents or field technicians to quickly diagnose the problem that is impacting the end-user and to remediate issues. Interactions with LLMs can be broadly classified into three categories:

- *Conversational interactions*: These are the most common type of interactions that people have with ChatGPT or Gemini where the interactions are usually a series of prompts and responses. The responses are derived from the (static) data used during the pre-training process and therefore cannot be expected to be up to date.
- *Transactional interactions*: These interactions leverage the data specific to the application which was not used in the initial pre-training phase; for example, on an e-commerce website the user can go through a series of steps to return a merchandise. Leveraging the local application-specific dynamic data allows for the support of several use-cases, but the flow of the interaction is usually pre-determined and is meant to solve only a set of specific set of common interactions.
- *Interactive inferencing*: This is a sophisticated combination of the previous two types of interactions where the user is not simply restricted to follow pre-defined flow but can interact with the LLM using natural language questions. The application back-end is augmented to leverage application specific data and the actions or transactions invoked by the application are inferred from the user queries. One or more agents are used to break down the actions required into smaller components, perform the required operations and then aggregate the results to accomplish the task requested by the user. This type of interaction allows the user to interactively work with the LLM and the application-specific data to solve complex problems.

In the following sections we will start off with the description of how to build applications for conversational interactions and then address the challenges associated with developing more complex solutions required to support transactional interactions and interactive inferencing.

3. LLM Performance Improvement Techniques

Foundation LLMs are mainly *autoregressive* models that are pre-trained on a massive corpus of text mainly to predict the next word or a set of words until completion. These models are pre-trained using a self-supervised learning approach by providing as input the beginning of a text sample and tasked to predict the next word. The target word or the ground-truth happens to be the actual next word in the input text sample. For example, the most recent Llama-3 model from Meta [meta-llama3-2024] has a vocabulary of 128K tokens and was pre-trained on 3T (trillion) tokens of data that were all collected from publicly available resources. However, there is no guarantee that these foundation models have been trained on data that is specific to certain domains like medical, legal, or broadband communications. Therefore, there is always a need to adapt these foundation models to perform specific tasks such as translation or review rating, or to enhance them on a knowledge base that is specific to a domain.

Prompt engineering or crafting prompts in an appropriate fashion can be used to get more relevant answers from LLMs. While zero-shot prompting, where the user simply includes a question in the prompt for the LLM, may work reasonably well for models with large number of parameters, smaller LLMs usually do not perform well with simple prompting. Few-shot prompts. whereby a series of examples are provided as contexts. usually work much better in steering the model to perform the task that the user is interested in. For queries involving complex logical reasoning chain-of-thought (CoT) prompts have been shown to produce better results. CoT prompting involves few-shot prompts where the exemplars include a series of logical reasoning steps [Wei 2022]. Zero-shot CoT prompts have been shown to be effective by adding the phrase “let’s think step-by-step” to the prompt. Few-shot and CoT can be combined where the user provides examples of few questions with explanation of how the answers were derived in a step-by-step fashion followed by the actual question.

While prompt engineering can be an effective tool in improving an LLMs performance, it cannot be relied upon when building a virtual assistant tool as it is not reasonable to assume that the end user of that virtual assistant will be familiar with the concepts of prompt engineering or knowledgeable enough to craft effective queries. Therefore, it is necessary to explore available options to customize foundation

LLMs for our specific applications. There are two broad approaches to adapting foundation LLMs, namely, Fine-tuning and RAG.

3.1. Fine-tuning

A significant amount of research work has gone into fine-tuning pre-trained LLMs to improve their performance and generalization to new tasks or applications on a domain-specific dataset. Fine-tuning is also termed as instruction fine tuning or supervised fine tuning, is a strictly supervised learning process. It has been shown that fine-tuning can not only significantly improve the performance of an LLM but can also enable fine-tuned small models to perform better than very large pre-trained models that are not fine-tuned. [chung-google-2022] shows results of fine-tuning that can scale to a large number of tasks and to models of different sizes. The fine-tuning process typically consists of the following steps:

- Identify the pre-trained model to be fine-tuned. This selection can be based on the size of the original model as well as the data that has been used to train the original model.
- Collect training data (for example, a list of prompts and responses) that is appropriate for the task or domain. Note that while the amount of data required for pre-training LLMs is enormous, the data required for fine-tuning is significantly smaller and more manageable. This training data can be obtained from publicly available sources or needs to be curated with human resources.
- While pre-training models require a large amount of computational resources, for example, Llama-3 was trained on a custom cluster of more than 24K GPUs, fine-tuning can be performed on relatively modest GPU resources, time, and budget.
- Since the original pre-trained models can have parameters in the range of 1B to more than 100B, it is not practical to update all model parameters during fine-tuning with a limited dataset. Parameter Efficient Fine Tuning (PEFT) algorithms are commonly used in the fine-tuning training process. Algorithms such as LoRA (low ranked adaptation) or QLoRA (quantized LoRA) do not update the original model parameters directly but generate a low-order matrix of parameters that is trained using the new training data. This low-order matrix is then summed with the original model parameters. With the PEFT algorithms the number of model parameters that are updated can be as little as less than 1% of the original pre-trained model parameters.

3.2. Retrieval Augmented Generation (RAG)

In the fine-tuning approach discussed in the previous section, the parameters of the LLMs are modified during the fine-tuning process based on the training data. Another approach that is commonly used to improve the LLM's performance is RAG whereby the model parameters are left unchanged, but additional relevant contextual information is provided to the LLM that can significantly improve the performance on domain specific applications. While there are advanced RAG approaches, discussion in this paper will be restricted to the basic RAG approach which employs the following steps:

1. Identify the domain specific data or more recent data that the model has not been already pre-trained on. This data can be in the form of text files, pdfs, html documents and more.
2. Textual data from these documents are split into small pieces using text splitters and then combined to form chunks. Choices of text splitters and chunk sizes can have implication on the performance of the RAG system as presented later in this paper.
3. Contiguous sections of textual chunks are then mapped into a vector space using an embedding model. These vectors are indexed and stored in a vector database such as Pinecone, ChromaDB, or FAISS. Choice of the embedding model has implications on both the retrieval performance and other factors such as latency.

4. When the user inputs a query to the LLM, the query is transformed to a vector using the same chunking and embedded models and passed to the vector database which returns the top K matches (Top-K) from comparing the query to the stored data.
5. The return Top-K matches are used as contexts and integrated with the user query to pass onto the LLM.
6. Steps 1 through 3 can be repeated as required to incrementally index additional data that becomes available as the RAG approach does not modify the LLM model parameters at all.

Figure 1 below shows the steps involved in indexing the data from the domain specific knowledge base and the response generation augmented by retrieved contexts.

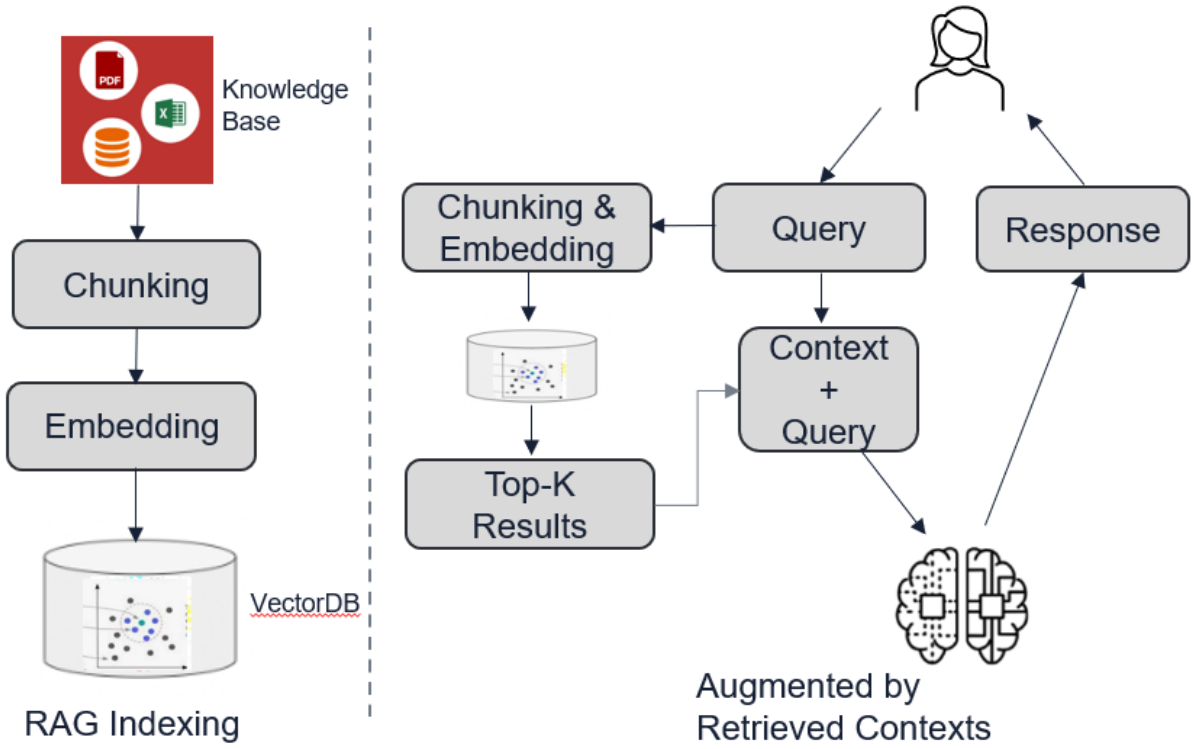

Figure 1 - RAG processing pipeline and architecture.

3.3. Comparing Fine-tuning and RAG

It should be noted that the two approaches presented above are not mutually exclusive. There could be scenarios where using a fine-tuned LLM model with RAG is the more appropriate option. In fact, there are several fine-tuned models available (usually prefixed with FLAN standing for fine-tuned-language-models, like FLAN T5, FLAN-PaLM [flan-2021]) that can be used along with RAG. Fine-tuning attempts to adapt or tweak the model by changing the model parameters, therefore what is learnt through the supervised learning process is baked into the model, whereas RAG leaves the model parameters untouched while attempting to provide better contexts. Because of this, some researchers tend to refer to fine-tuning as changing the "long-term" memory of the model while RAG is akin to improving the "short-term" memory of the model. The following table summarizes the main differences between the two approaches that should be taken into consideration while comparing the fine-tuning versus the RAG approach.

Table 1 - Comparison of RAG and Fine-tuning approaches to LLM optimization

	Fine-tuning	RAG
Ability to adapt the model to new use-cases and unseen data	Yes	Yes
Handling dynamic and incremental data	More difficult	Easy
Difficulty in curating training data	Higher	Lower
Reducing hallucinations	Yes	Yes
Latency in inference generation	Normal	Slightly higher due to context generation
Cost of inference generation	Normal	Slightly higher due to larger context
Transparency	Less	High
Technical expertise needed	Higher	Lower

4. Evaluating RAG Performance

Evaluating the performance of LLMs using objective measures has turned out to be a difficult task for both researchers and practitioners deploying LLM based applications. It stems from the fact that semantic understanding of text is highly subjective. However, there are several objective metrics developed by natural language processing (NLP) community that can be used here. Another challenge with RAG/LLM evaluation is understanding the performance impact of various parameters associated with operations such as chunking, embedding, etc., in the indexing process and the parameters associated with the LLM in generating the response. This problem is not unique to RAG or LLM, but similar problems exist in deep neural networks where it is often difficult to pinpoint which hidden layer nodes heavily contribute to the final classification or regression performance of these networks.

We have taken a two-step approach to evaluate RAG performance:

- End-to-end performance of response generation of LLM by comparing the responses with a ground-truth reference response crafted by a human expert.
- Relevancy and efficacy of contexts generated by RAG semantic search from the vector database index

4.1. End-to-end Performance Evaluation

For a meaningful end-to-end objective performance evaluation, the following are required:

- A set of input queries, preferably with some of the queries coming from the knowledge base information that has been indexed and some of the queries from external sources.
- Reference (or expected) responses for each input query. Reference responses can be obtained from an existing dataset, if available, manually crafted by an expert or generated automatically by another LLM. Reference responses are also termed as "ground truth" in this document.
- One or more objective metrics to compare the reference response text with the actual generated response text for each query.

We created a list of twenty-seven queries to study the end-to-end performance. As mentioned above a subset of these questions originated from the indexed knowledge base, but many of the questions originated from other sources like SCTE research papers and on-line FAQs related to DOCSIS® networks, proactive network maintenance (PNM) and cable access architecture. For queries extracted from the knowledge base, the reference responses were also derived from the knowledge base and in some cases as summarized by a human expert. For queries that originated from external sources, the reference responses were crafted by a human expert.

As an example, the following is a query and reference response used in our evaluation. All the queries used in our evaluation are pertinent to DOCSIS networks, PNM or cable network architecture:

> **Query**: *What is Modulation Error Ratio and how is it used in DOCSIS cable networks?*
>
> **Reference Response:** *Modulation error ratio (MER) measures signal power versus constellation error magnitude. Constellation error magnitude encompasses all impairments that can degrade the digital signal, not just white noise. MER measures the received symbol vector and calculates the difference between it and the ideal signal vector. The power of the error vectors is averaged over time and can be viewed.*

We evaluated several objective measures from NLP literature in the initial phase of the evaluation for the sake of completeness. The measures used are [lin-2004] [lavie-2005]:

- BLUE – a measure of precision
- ROUGE – a measure of recall
- METEOR – a measure that combines precision and recall
- Cosine Similarity – a measure that compares two vectors

We also had a human expert evaluate the responses and attach a *subjective* score based on how closely the responses match with the references. Based on observations the BLUE and ROUGE metrics capture only one aspect (either precision or recall) of the responses and therefore not very appropriate. Since the METEOR metric combines both precision and recall, it represents a more balanced comparison of the responses and references. While BLUE, ROUGE, and METEOR metrics are computed by comparing words (or n-grams) in the responses against the references, the Cosine similarity maps the responses to a multi-dimensional vector using an embedding and then compares the similarity to the references using vector matching. Hence Cosine similarity is expected to capture more of the semantic meaning of the sentences. It should be noted that the objective measures will vary slightly from run to run for the same input queries as the responses generated by LLMs are not the same for successive runs. Depending on the configuration of the hyperparameters of the LLM (like the *temperature*) there can be some amount of variability in the generated responses on successive runs.

Figure 2 below shows the METEOR and Cosine Similarity computed for all the twenty-seven queries. As it can be seen from the plot, these two metrics have a reasonable amount of correlation. Note that both

these metrics fall in the range of [0,1], so the Cosine similarity, in general, generates scores that are closer to 1 compared to the METEOR score.

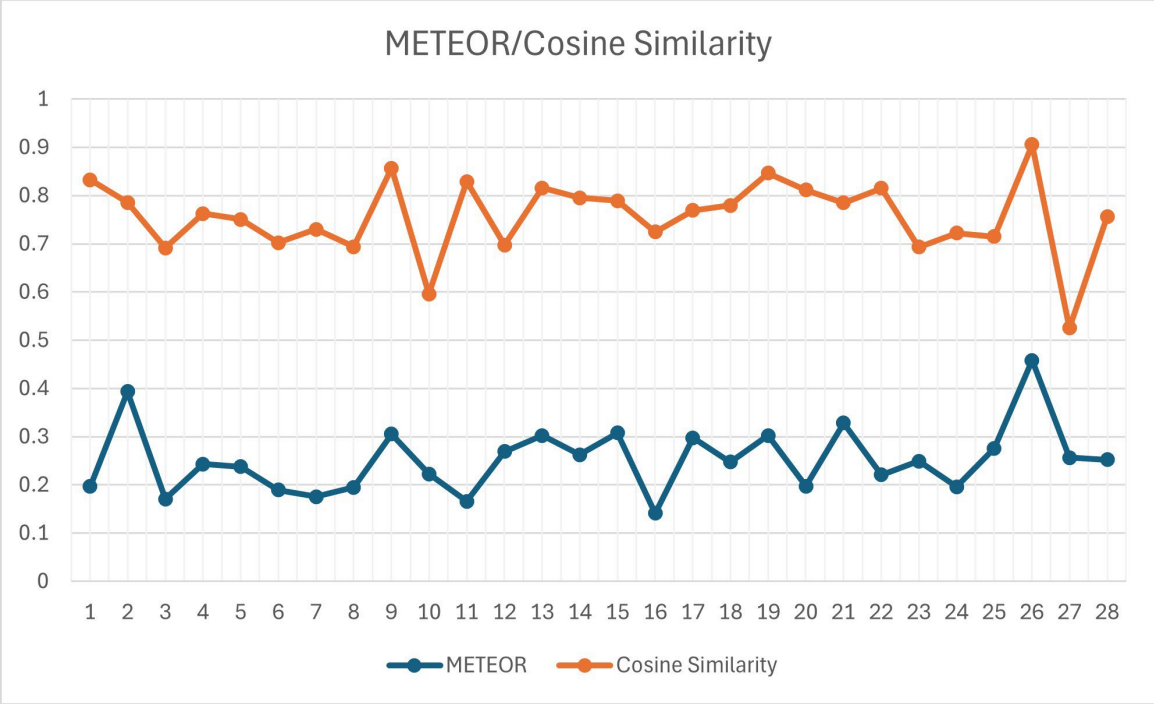

Figure 2 - METEOR and Cosine similarity computed for the 27 test queries

End-to-end performance evaluation is a useful tool to measure the performance of the whole system and monitor the changes in performance over multiple product release cycles. The biggest drawback of this end-to-end performance evaluation is the lack of transparency and visibility into the problems and identifying areas that need to be improved.

There are a number of choices to be made and parameters to be selected in creating the vector index. For example, creating the chunks from a set of long documents requires selection of an appropriate text splitter and the chunk size. Embedding these chunks into the vector space requires the selection of an embedding model. Finally, one needs to choose the number of chunks retrieved from the semantic search to be included in the context for the LLM.

In addition, there are a handful of parameters to be configured on the LLM such as the *temperature*, *top_k*, and *top_p* to control the randomness of the output. Setting the temperature to a value of zero makes the output more deterministic by forcing the LLM to pick the most probable next word or token, while setting it to a large value allows the output to be more variable. Parameters *max_new_tokens* limit the verbosity of the output and the *repetition_penalty* prevents the repetition of same words.

The lack of visibility and transparency of the end-to-end evaluation methodologies makes it difficult, if not impossible, to judiciously choose the aforementioned parameters. In the next section, we will describe certain metrics that are better suited for the selection of the parameters associated with vector indexing and semantic search.

4.2. RAG Context Efficacy and Relevancy

Since the main component of RAG is the indexing of the domain-specific knowledge base and retrieving the top matches from the index for a given query, it behooves to measure the relevancy of the retrieved contexts for a given query and the efficacy of the contexts in the response generated by the LLM. Figure below shows the use-case where for all each query used during evaluation there is a corresponding expected response or ground truth. This expected response may come from an existing dataset, generated by a human expert, or in some cases may have been generated by another LLM. As shown in Figure 3, the end-to-end performance evaluation described in the previous section simply compares the ground-truth and the actual response for each test query. On the other hand, evaluating context efficacy requires comparing the retrieved contexts to the generated response, ground truth and in some cases with the query itself as shown by the dotted arrows.

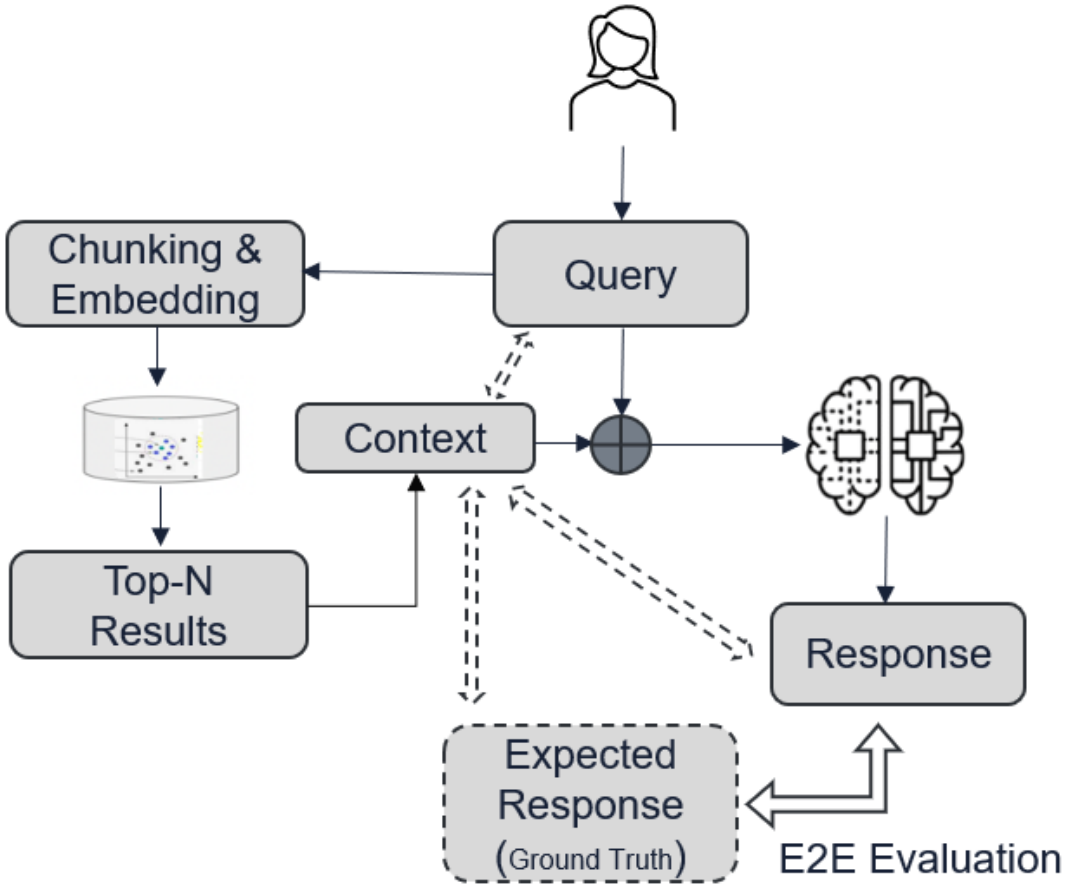

Figure 3 - RAG Evaluation system

Following is an example of a user query and the top two chunks retrieved from the vector store which are then passed as context to the LLM along with the query and the resulting response generated by the LLM. Frameworks presented below were used to evaluate relevancy the retrieved context chunks are to the query, generated response and the expected response can be attributed to the context chunks.

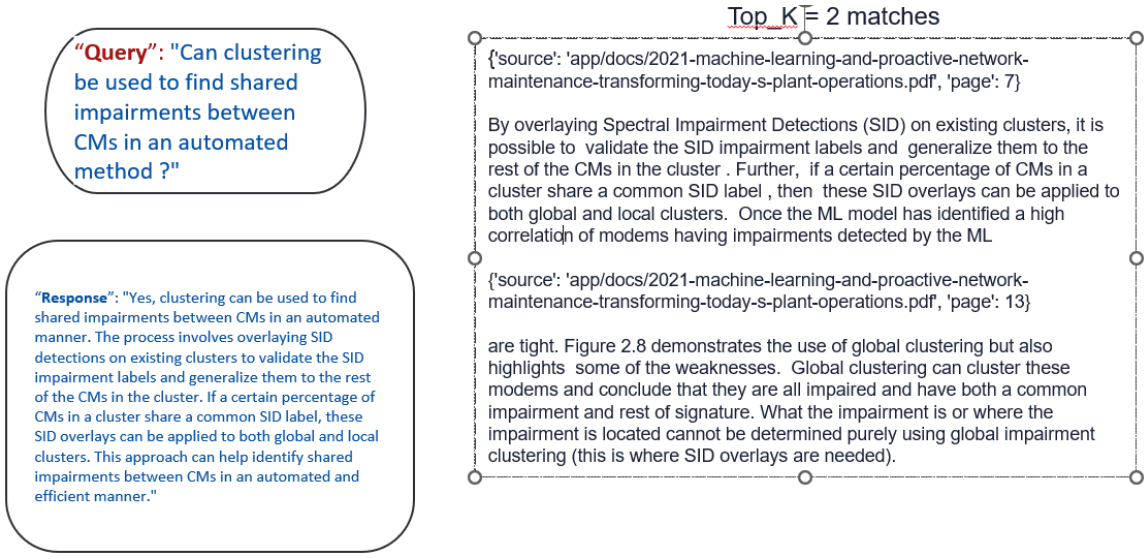

Figure 4 - Sample query, top two retrieved contexts, and the responses generated by the LLM

There are a few open-source and commercial tools available for evaluating the RAG pipeline. They all use different, but somewhat similar metrics and in most cases use another LLM (mostly chat-gpt) in generation of these metrics. A word of caution is that different frameworks use the same name for a metric, but the underlying implementations are somewhat different; for example, the *Answer Relevancy* is computed very differently in RAGAS compared to DeepEval. Therefore, for a successful evaluation of the RAG pipeline, it is necessary to inspect the implementation details of the metrics generated by these frameworks and to assess how relevant these metrics are for a given application. We evaluated the following open-source tools:

- RAGAS
- Arthur Bench
- DeepEval

Precision and Recall metrics have been used to evaluate discrimination functions in statistical hypothesis testing for several decades, where precision measures the accuracy of discrimination function (out of all the generated positive hypotheses how many are truly positive) and the recall measures the completeness (out of all the true positives in the ground truth how many were correctly flagged by the discrimination function). The frameworks presented above use variants of the precision and recall metrics for the context of RAG and LLMs. Some of the metrics from the RAGAS framework that we found informative are listed below and note that all of this metrics are in the range of 0 to 1 with a value of 1 being the best:

- **Context Precision**: To compute this metric certain statements/claims are identified from the ground truth and the retrieved chunks are evaluated to see if a given chunk is relevant to those statements. This metric also incorporates the rank of the relevant chunks, i.e., the relevant chunks should have a higher rank.
- **Context Recall**: Context recall is a metric that is computed using the ratio of the number of statements (in the ground truth) that can be attributed to the context to the total number of statements.
- **Faithfulness**: This is similar to the Context Recall, but the statements/claims are generated not from the ground truth response but from the actual generated response.

- **Answer Relevancy**: This metric is an assessment of how relevant the generated response is to the given prompt or query.

We used the metrics above to drive the appropriate selection of various attributes and parameters in the RAG system, namely, text splitters, chunk size, embedding, number of chunks (Top-K) in the context, etc. The following table shows the results for one of the parameters, Top-K, the number of chunks retrieved from the vector database. In an ideal scenario, there will be one value of K that optimizes all the metrics, but as can be seen from the table below that's hard to achieve. Boxes highlighted in green represent the largest (or close to the largest) values for each metric. That's not totally surprising as in classical statistics precisions and recall are divergent metrics and algorithms choose the best trade-off between precision and recall. Based on the table below, it is easy conclude that a choice of top_K value of 4 or 5 is a reasonable choice.

Table 2 - RAGAS metrics computed for various Top-K chunks retrieved from the vector datastore

Top-K	Context precision	Context recall	Faithfulness	Answer relevancy
2	0.83	0.94	0.58	0.78
3	1.00	0.83	0.67	0.92
4	1.00	1.00	0.96	0.93
5	0.99	1.00	0.95	0.94

We have used a similar evaluation methodology to make optimal choices for other parameters such as the embedding model, chunk size for the RAG indexing and various other parameters associated with the LLM.

5. Leveraging Dynamic Data

Discussions in the previous sections were mainly centered around indexing documents from a knowledge base and using semantic vector search to create contexts to add to the queries dispatched to the LLM. This approach can be used to build virtual assistants to enable "Conversational Interactions" as described in Section 2. The ability to update information or documents indexed in the vector store at any time without impacting the rest of the LLM pipeline is one of the major advantages of RAG. In practice, however, the indexed information is only updated periodically or may not leverage other data collected from the network devices in real-time or near real-time. Network operators collect diagnostic and operational data from devices such as CMTSs, CMs, routers and switches; some of these data is collected using legacy polling techniques such as SNMP whereas some devices support more recent model-based telemetry using IEEE Yang Push or OpenConfig streaming telemetry. Relevant data may also be available from third-party resources like outage reporting websites, weather sites, web search, etc.

LLM RAG approaches can be seamlessly augmented to leverage additional data not indexed in the vector store. Frameworks such as LangChain use an interface called *Retriever* that is designed to return documents for any unstructured query. Context chunks used in RAG are pulled from the vector store using this Retriever interface. This same interface can be used to retrieve documents from other APIs and supports building *Custom Retrievers*. Custom retrievers are essentially API endpoints that can access additional data sources, perform necessary operations and return results in the form of documents. Any

required business logic can be implemented in these custom retrievers. Such implementations can support the "Transactional Interactions" described in Section 2 of this paper.

LangChain can also be used integrate with a number of third-party retrieval services. A popular one is the Tavily search API which is a search engine specifically optimized for LLMs and RAG provided efficient and quick search results. Figure below shows the generalized use of the Retriever interface that can be used with semantic search of vector DB, custom APIs that operate on the proprietary private data and/or third-party APIs that operate on public data sources.

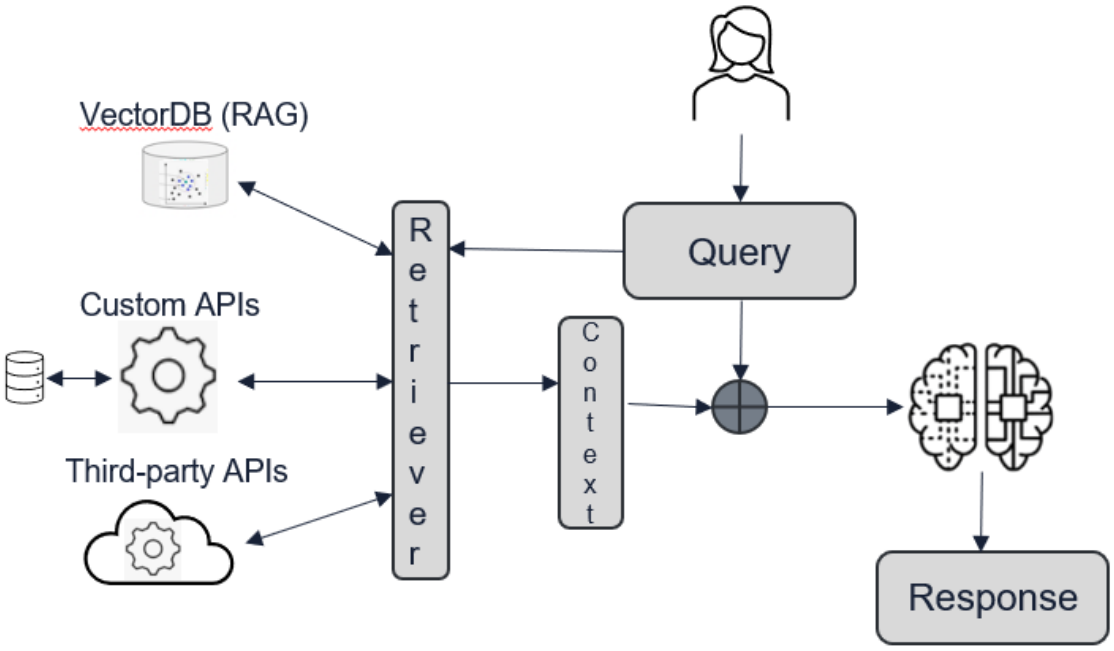

Figure 5 - Generalized Retriever approach to use custom and third-party APIs

To support user interactions with LLM to accomplish "Interactive Inferencing" described in Section 2 requires the use of *agents* or Agentic-RAG. A complete discussion of Agentic-RAG is beyond the scope of this paper and hopefully will be addressed in the future. Agents are software components that can perform more sophisticated analysis of the query to break down the problem into sub-components and can be endowed with a plan formulation to solve problems associated with the query. Agents are also equipped with a set of application-specific tools and can be trained to use the appropriate subset of tools based on the query. Depending on the configuration, agents can use their short-term or long-term memory not just to recall past queries but the results of interactions of the past queries.

6. Conclusion

"RAG systems are easy to build but are very difficult to master" – this was an online quote that the author ran into couple of years ago and this is indeed true. While standing up a RAG system with a LLM is a relatively easy task, gaining a deep understanding to optimize the system and to improve failure scenarios is considerably more complex as discussed in this paper. Designers of these applications need to painstakingly understand the trade-off associated with various choices in the building blocks of these systems. "Drowning in data but gasping for insights" is true in many industries today. With the amount of data collected by the broadband communication providers increasing by an order of magnitude in the last few years, leveraging machine learning and AI is indispensable to extract insights from this trove of

data. Interpreting data associated with capacity planning, proactive network maintenance, and network optimization and drawing meaningful conclusions are still tasks that can be performed only by a few experts in most organizations. Incorporating AI agents that can perform interactive inferencing into the workflow can immensely help in broadening that expertise to a larger pool of engineering and operations talent.

Abbreviations

BLUE	Bilingual Evaluation Understudy
CoT	Chain-of-thought
FLAN	Finetuned LAnguage Net
GPU	Graphics Processing Unit
LLM	Large Language Model
MER	Modulation Error Ratio
METEOR	Metric for Evaluation of Translation with Explicit ORdering
NLP	Natural Language Processing
PNM	Proactive Network Maintenance
RAG	Retrieval Augmented Generation
ROUGE	Recall-Oriented Understudy for Gisting Evaluation

Bibliography & References

[ulm-2019] "Traffic Engineering in a Fiber Deep Gigabit World", https://www.nctatechnicalpapers.com/Paper/2017/2017-traffic-engineering-in-a-fiber-deep-gigabit-world
[volpe-2021] "Machine Learning and Proactive Network Maintenance: Transforming Today's Plant Operations", https://www.nctatechnicalpapers.com/Paper/2021/2021-machine-learning-and-proactive-network-maintenance-transforming-today-s-plant-operations
[righetti-2023] "Machine Learning Model for Customer Claim Prediction in HFC Subscribers", https://www.nctatechnicalpapers.com/Paper/2023/3578_Righetti_5242_paper
[vaswani-2017] "Attention is all you need", https://arxiv.org/abs/1706.03762
[meta-llama3] "Introducing Meta Llama 3: The most capable openly available LLM to date", https://ai.meta.com/blog/meta-llama-3/
[wei-2022] "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models", https://arxiv.org/abs/2201.11903.
[chung-2022] "Scaling Instruction-Finetuned Language Models", https://arxiv.org/abs/2210.11416
[flan-2021] "Finetuned Language Models Are Zero-Shot Learners", https://arxiv.org/abs/2109.01652
[lin-2004] "ROUGE: A Package for Automatic Evaluation of Summaries", https://aclanthology.org/W04-1013.pdf
[lavie-2005] "Automatic Machine Translation Evaluation System", https://www.cs.cmu.edu/~alavie/METEOR/