Supercharging Document Composition with Generative AI: A Secure, Custom Retrieval-Augmented Generation Approach

Andre Chen
Data and AI Discovery Lab
Accenture Federal Services
Washington, D.C., United States
andre.chen@afs.com

Sieu Tran
Data and AI Discovery Lab
Accenture Federal Services
Washington, D.C., United States
sieu.tran@afs.com

***Abstract*—Recent advancements in Retrieval Augmented Generation (RAG) provide opportunities for more efficient composition of long-form, domain-specific documents. However, few user-friendly applications leverage RAG for this specific use case. Additionally, existing RAG frameworks reliant on cloud-based, closed-source solutions pose challenges such as transparency and data privacy concerns. To address this gap, we introduce a full-stack application for automated document drafting, offering either proprietary (GPT-3.5-Turbo) or open-source (Falcon-RW-1B-Chat) Large Language Models (LLMs) for document writing and a secure, self-hosted search index (Elasticsearch) for knowledge retrieval. Users interact via a simple UI in which they submit a request with a desired topic and document type and receive a well-researched document that conforms to specific formatting and structural requirements. Given a user request, our document search pipeline employs open-source encoder models from SentenceTransformers for vector search and semantic re-ranking, then passes relevant knowledge sources to the LLM as context for document composition. To orchestrate and modularize our application backend, we use Haystack, an advanced machine learning pipeline builder, and FastAPI, which enables us to deploy components of our pipeline as independent services. Our experiments demonstrate that, using highly customized, self-hosted components, we can achieve document generation quality comparable to that of fully online RAG pipelines. This enables us to provision different application versions to accommodate different cost, scale, and data security preferences.**

***Index Terms*—generative AI, LLMs, retrieval-augmented generation, hybrid search, semantic rerank, ML pipeline, vector databases**

I. Introduction

Drafting domain-specific documents is often an inefficient and redundant endeavor, particularly within large enterprises. Such documents often involve research and consolidation across different departments to pool the requisite information, followed by the labor-intensive task of writing and revising content. This procedure is time-consuming and suboptimal for three key reasons: (1) different departments create variations of zero-stage boilerplate content instead of referencing consistent, gold standard examples, (2) writers typically cannot search exhaustively through all relevant knowledge sources, leading to wasted time recreating content that already exists elsewhere, and (3) communication and knowledge-sharing between relevant departments becomes increasingly difficult as organizations grow in size and complexity.

To alleviate these pain points, we present an application that automatically searches relevant sources to produce complete and cohesive first drafts, assisting the user in both the research and writing process. This approach draws upon recent advancements in retrieval-augmented generation (RAG), which combines large language models (LLMs) and efficient storage and retrieval methods to increase accuracy and reliability on knowledge-intensive NLP tasks [1, 2, 3]. The RAG framework has demonstrated success in reducing hallucinations and embedding domain-specific expertise in LLMs without computationally expensive retraining. This capability is particularly desirable when automating natural language tasks that require background knowledge from everchanging sets of internal documents.

Our work combines the RAG framework with intelligent application design for section-by-section document composition. For document storage and retrieval, we use Elasticsearch [4], which enables keyword filtering and vector search over documents using open-source encoder models, such as those introduced in SentenceTransformers [5]. Elasticsearch is advantageous for our use case in that it offers horizontal scalability and efficient storage and retrieval of unstructured text documents through its distributed NoSQL storage format. Furthermore, it can be easily configured to use different advanced retrieval strategies such as vector search, hybrid search, and semantic re-rank with the help of external libraries (e.g. Haystack).

Following the retrieval step, we use a BERT-large encoder model to perform extractive summarization over our input context, then offer two options for content generation models to serve different cost and data security preferences. The first option uses OpenAI's GPT-3.5-Turbo model through the Azure OpenAI service, which is more performant for long-form text generation but requires data transmission through an online API and incurs cost based on usage. The second option uses Falcon-RW-1B-Chat (*ericzzz/falcon-rw-1b-chat* on

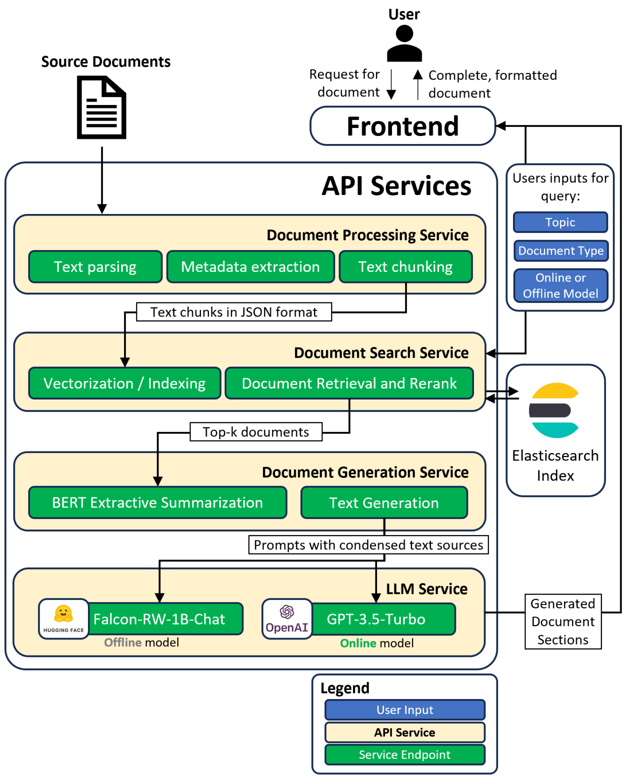

Fig. 1. Solution Architecture and Information Flow

Huggingface), a conversational instruction-tuned version of the original Falcon LLM trained on the RefinedWeb dataset [6, 7]. This model is open-source and downloadable for self-hosting, offering users an alternative option from a cost and data security perspective.

To orchestrate our RAG pipeline, we use Haystack [8], a Python framework by deepset AI for building moduluar and customizable machine learning pipelines. Haystack benefits our solution by providing chains of highly configurable building blocks (nodes) that implement key processing tasks. Within our system, we connect its Preprocessor, DocumentStore, and Retriever nodes together to build our ingest and retrieval pipeline. These nodes support a wide array of database technologies, encoder models, and retrieval approaches, enabling rapid experimentation with different components and pipeline configurations to optimize the RAG system.

In addition to Haystack, we also use FastAPI with CORS middleware to provision each RAG component as a separate API service with strict input schemas enforced through Pydantic data models. This is advantageous from both an engineering and cost-scaling perspective: it enables targeted testing of any component of the pipeline without the need to run preceding pipeline nodes, and it allows us to tailor the type and scale of compute resources to each pipeline component. Figure 1 depicts our application architecture, including API services and information flow.

Through our work, we demonstrate that the challenges of long-form, domain-specific document composition can be addressed by leveraging three commonly observed document characteristics: (1) repetitive structure and content style, (2) repurposed content from previous similar work, and (3) discrete sections that each address a specific topic or requirement. We also present a series of experiments evaluating different RAG configurations and comparing selected system components against similar available alternatives. From these experiments we optimize design choices across the RAG pipeline for our use case, from data retrieval configuration to language model selection and prompt design. Furthermore, we show that, using smaller open-source LLMs, we can achieve comparable outputs to those from larger proprietary LLMs (e.g. GPT-3.5-Turbo) on our generation task.

II. Related Work

Prior to the original 2021 RAG paper by Lewis et al., related research in retrieval-augmented techniques had driven significant improvements in LLM accuracy and hallucination rate, establishing the information retrieval step as a critical component in knowledge-intensive language tasks [9, 10, 11, 12]. However, Lewis et al. further differentiated the RAG framework from these approaches in that their performance gains could be realized on everchanging and arbitrarily large knowledge sources without computationally expensive retraining procedures [1].

Recent efforts have since endeavored to use RAG to generate more sophisticated outputs with higher fidelity. Notably, Liu et al. characterized how longer prompts (i.e. those containing knowledge sources as context) are attended to by modern LLMs for use cases involving information access, while Chen et al. provided concrete directions for use of retrieval-augmented generation in long-form question answering, a key enabler of retrieval-augmented document writing [13, 14]. These studies present two key challenges in retrieval-augmented generation of long-form text: (1) language models do not robustly make use of information in extremely long input contexts, tending to favor content at the beginning and end, and (2) the order of retrieved knowledge sources, which is difficult to control in the retrieval phase, impacts the order of information presented in the output.

Our approach to document composition addresses these key challenges by combining the RAG framework with ideas from related work leveraging smaller models on knowledge-intensive tasks. Specifically, Li et al. present an approach to evidence-based text generation that includes extractive, followed by abstractive, summarization of source documents [15]. Similarly, we use the BERT-large encoder model for an extractive summarization step prior to “abstractive” generation with a larger downstream LLM, which is useful in addressing challenges (1) and (2). In particular, it condenses the context based on how semantically similar each sentence is to the core meaning (centroid sentence vector) of the broader passage, allowing us to achieve a more natural ordering of ideas for the abstractive generation step [16].

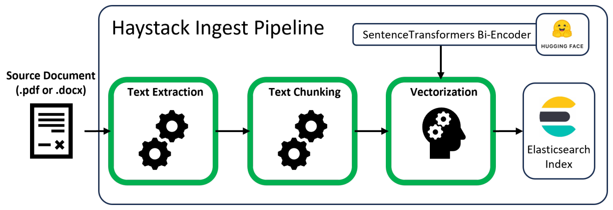

Fig. 2. Document preprocessing and upload pipeline

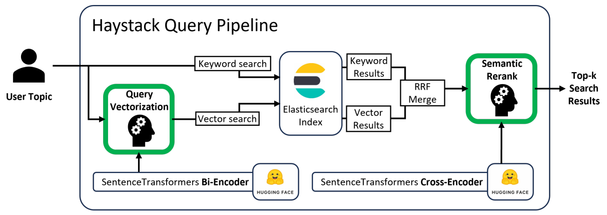

Fig. 3. Retrieval-augment generation framework

III. Methodology

The proposed application for document composition can be separated into three broad components: information preprocessing/ingest, information retrieval, and document generation. Figure 1 depicts the flow of data through these components. The application receives as user input a series of key topics to write towards and a desired document type. The document type is passed directly to the document generation step, while the topics are run as queries against our vector database (Elasticsearch) to retrieve relevant knowledge sources. Knowledge sources are then condensed using extractive summarization and rewritten using section-specific prompts. We describe these processes in more detail below.

A. Document Preprocessing and Ingest

To initialize our information database, we first apply the following transformations to render input knowledge sources (PDF and Microsoft Word documents) in usable format (Figure 2):

1) **Conversion** of input knowledge sources (PDF and Microsoft Word documents) to raw text.
2) **Chunking** of extracted texts into fixed-length segments. We find that optimal chunk size varies by use case and retrieval methodology, and report an analysis on chunk size in the Results section for an example use case.
3) **Augmentation** of each chunk with additional metadata, including the year the source document was written, the source document type, etc.
4) **Encoding** of each chunk's text and metadata into dense 768-dimensional vector representations to enable vector search. We use the *all-mpnet-base-v2* model, a fine-tuned version of Microsoft's MPNet model [17] as our encoder given its extensive fine-tuning on sentence pairs. We hypothesized that contrastive fine-tuning over sentence pairs would make the *all-mpnet-base-v2* model a strong similarity search-based retriever model.

We load processed text chunks with their corresponding embeddings into our Elasticsearch index, which we leverage for both keyword-based filtering and vector search in the information retrieval phase.

B. Information Retrieval

User queries are processed through a two-phase retrieval and rerank approach to retrieve relevant document chunks (Figure 3), which has shown improved performance over single-phase retrieval-only methods in the literature [18, 19, 20, 21] and in our own experiments.

The retrieval phase employs a hybrid sparse/dense encoding search involving both keyword search and vector search. Keyword search employs the BM25 algorithm [22] to score and rank potential query matches. To perform vector search, we use a bi-encoder retriever model (*all-mpnet-base-v2*) to map query text to the same 768-dimensional embedding space as our document chunks, then retrieve top-k most similar document chunks using vector cosine similarity as our scoring metric. Rankings from both scoring methodologies are then combined using reciprocal rank fusion (RRF) [23], from which we obtain top-k results to pass to the rerank phase. For our example use case, which we discuss in the Experiments section, we select a top-k of 25 based on experimentation with several different values, taking ranking scores and processing time into account.

In the final re-rank phase, results obtained in retrieval are reordered using a separate cross-encoder model (*ms-marco-MiniLM-L-12-v2*) and truncated to the top 10 most relevant documents to pass to the LLM context. In contrast to the retrieval phase, in which document scores are based on cosine similarity between independently computed source document embeddings and query embeddings, the re-ranking cross-encoder performs a computationally expensive inference step *pairwise* between the query and each source document at query time. Thus, the scoring mechanism of the reranking step is more time-intensive but typically provides a more reliable measure of semantic similarity between texts, making it suitable as a refinement step after the initial retrieval phase.

C. Document Generation

Prior to the final text generation step, we apply extractive summarization to retrieved documents using the *bert-extractive-summarizer* package [16], which works by separating the input text into sentences, encoding sentences into vectors using BERT-large [24], and retaining the top-k most similar vectors to the centroid of the sentence vectors. This step ensures the provided content fits within typical LLM context windows while retaining ideas that align most closely to the central theme of the retrieved text.

The desired document type specified by the user is mapped to a set of pre-coded sections (e.g. Title, Highlights, Body, Conclusion sections for a news summary report document type), each with a bespoke task prompt with instructions on how to construct the specific section. For the purposes of demonstration, we focus on the news summary report document type in experimentation and evaluation, but note that this approach can be generalized to other document types that also have a well-established format and structure.

For each section, the application composes a message log consisting of the user-specified document topic, the retrieved and summarized knowledge sources, and the section task prompt. The system then sends this message log via API request to the application's LLM service, specifying additional model parameters such as temperature and max_tokens in the request body. This service then routes the request to the offline Falcon-RW-1B-Chat model or the Azure OpenAI chat completion API depending on user preferences.

Once all sections are generated for the user topic, the system consolidates all generated texts into a single document object organized by section to be returned to the user.

IV. Experiments

Our solution is easily configurable across the RAG pipeline to achieve optimal performance for different document types. To illustrate this, we present a set of targeted experiments conducted in order to evaluate and optimize the performance of our solution for an example use case. Specifically, we demonstrate how we optimize our system to generate summary reports based on news articles retrieved for a particular topic. We use a set of 30,000 news articles from the open-source CNN/Daily Mail dataset, encoded and uploaded to our Elasticsearch index, as our example knowledge source. We then conduct two key experiments to measure performance of our information retrieval and generation components respectively.

A. Retrieval Experiment

To build our retrieval evaluation set, we sample 150 documents from our CNN/Daily Mail source documents to create a set of query/document pairs. This enables us to measure the performance of our information retrieval system using mean reciprocal rank (MRR), which quantifies how often the correct document for a given query appears in the top-k ranked results retrieved. With respect to our configuration search space, we test 3 different retrieval methods: sparse (keyword search only), dense (vector search only), and hybrid (keyword and vector RRF). We also measure performance with and without a secondary re-rank step. Additionally, where the re-rank step is applied, we further test 4 different retrieval top-k values prior to re-rank (25, 50, 75, and 100). We present scores and processing times for each of these 15 configurations in the Results section.

In order to validate our choice of Elasticsearch as our information retrieval engine, we also compare its performance against the enterprise Azure AI Search service, which provides a managed document index that is widely popular due to its performance, flexibility, and ease of integration with OpenAI models for RAG use cases. Azure AI Search also offers sparse, dense, and hybrid retrieval settings with semantic re-rank, as well as OpenAI's proprietary *text-ada-embedding-002* model for text encoding. As such, we are able to make direct comparisons in performance across all configurations tested in Elasticsearch.

B. Generation Experiment

Our generation experiment achieves 2 goals: (1) qualitative evaluation of OpenAI's GPT-3.5-Turbo model's ability to generate relevant, coherent content adhereing to the formatting requirements of different document sections, and (2) identification of smaller, open-source models that achieve comparable qualitative performance to GPT-3.5-Turbo, allowing us to offer potentially lower-cost, offline alternatives that enable greater data privacy.

TABLE I
Section generation prompts by LLM

Model	Section	Prompt
Pegasus	Title	Single sentence title about the following article text: [retrieved context]
Pegasus	Highlights	List of 5 highlights from the following article text: [retrieved context]
Pegasus	Conclusion	Concluding paragraph about the following article text: [retrieved context]
Falcon	Title	Create a short, one sentence news report title relevant to the topic of [topic] from the following text: [text] Your title:
Falcon	Highlights	Extract a numbered list of 3 top highlights relevant to the topic of [topic] from the following text: [text] Your highlights:
Falcon	Conclusion	Write brief, 3-sentence concluding paragraph for a report on the topic of [topic] from the following text: [text] Your concluding paragraph:
GPT-3.5-Turbo	Title	Given the Raw Source Content below, create a short title for a news article summary report on the provided topic. ##Topic: [user topic] ##Raw Source Content: [retrieved context] Your output:
GPT-3.5-Turbo	Highlights	Given the Raw Source Content below, summarize the top 3 key highlights as a numbered list to address the provided topic. It should be high-level, one sentence per highlight. ##Topic: [user topic] ##Raw Source Content: [retrieved context] Your output:
GPT-3.5-Turbo	Conclusion	Given the Raw Source Content below, write a brief (maximum 3 sentences) concluding paragraph for a report addressing the provided topic. ##Topic: [user topic] ##Raw Source Content: [retrieved context] Your output:

Our evaluation set consists of five distinct topics for which we generate a news summary report using each model:

1) Natural Disasters on the West Coast
2) Political developments on Capitol Hill in the United States
3) Business developments in the healthcare industry
4) New football player contracts in the English Premier League
5) Updates on international celebrities

We input these topics as queries to our full end-to-end RAG pipeline, changing only the content-generating model in each experiment variation. As our alternative candidate models to GPT-3.5-Turbo, we test Google's Pegasus model fine-tuned on the CNN/Daily Mail dataset and Falcon-RW-1B-Chat, available from Huggingface as *google/pegasus-cnn_dailymail*

TABLE II
PERFORMANCE OF DOCUMENT RETRIEVAL SYSTEM UNDER DIFFERENT CONFIGURATIONS

Retrieval Mode	Reranker Applied	Retriever Top k	Ranker Top k	MRR@10	Avg. Query Time (s/query)	Index
Sparse (BM25)	False	10	–	0.725405	0.013673	Elasticsearch
Sparse (BM25)	True	25	10	0.775757	9.137162	Elasticsearch
Sparse (BM25)	True	50	10	0.781241	17.119178	Elasticsearch
Sparse (BM25)	True	75	10	0.780812	30.175269	Elasticsearch
Sparse (BM25)	True	100	10	0.782757	41.845906	Elasticsearch
Dense (Vector Search)	False	10	–	0.684310	0.142271	Elasticsearch
Dense (Vector Search)	True	25	10	0.793220	9.873796	Elasticsearch
Dense (Vector Search)	True	50	10	0.791093	19.650987	Elasticsearch
Dense (Vector Search)	True	75	10	0.786963	29.653426	Elasticsearch
Dense (Vector Search)	True	100	10	0.785852	38.971187	Elasticsearch
Hybrid (Dense / Vector RRF)	False	10	–	0.760000	0.084302	Elasticsearch
Hybrid (Dense / Vector RRF)	**True**	**25**	**10**	**0.796378**	**9.121859**	**Elasticsearch**
Hybrid (Dense / Vector RRF)	True	50	10	0.788934	18.695639	Elasticsearch
Hybrid (Dense / Vector RRF)	True	75	10	0.786638	32.127264	Elasticsearch
Hybrid (Dense / Vector RRF)	True	100	10	0.785852	42.526988	Elasticsearch
Sparse (BM25)	False	10	–	0.694849	0.818298	Azure AI Search
Sparse (BM25)	True	50	10	0.737870	0.935096	Azure AI Search
Dense (Vector Search)	False	10	–	0.749516	1.332826	Azure AI Search
Dense (Vector Search)	True	50	10	0.749516	1.233675	Azure AI Search
Hybrid (Dense / Vector RRF)	False	10	–	0.739786	1.670826	Azure AI Search
Hybrid (Dense / Vector RRF)	True	50	10	0.758886	1.816402	Azure AI Search

and *ericzzz/falcon-rw-1b-chat*, respectively. We deliberately select these two models for distinct reasons: the Pegasus model demonstrates the performance of a strong abstractive summarization model fine-tuned on the system's source document set, while the Falcon-RW-1B-Chat model demonstrates the capability of a much smaller (relative to GPT-3.5-Turbo), instruction-tuned chat model.

With respect to generating our news summary report, we test each candidate model's ability to produce title, highlights, and conclusion sections for each of our five topics. Table I shows the model-specific prompts for generating each section, which we tailored through iterative refinement. Together, these three sections allow us to evaluate each model's ability to robustly generate high-quality, accurate text outputs and follow formatting guidelines for a variety of length and content requirements.

V. RESULTS

A. Retrieval Experiment

We select optimal configurations for document retrieval primarily based on MRR@10 (Mean Reciprocal Rank computed at a top-*k* of 10), with processing time as a secondary metric. Table II shows MRR@10 computed over all 15 configurations tested on our Elasticsearch index, as well as the 6 configurations tested in Azure AI Search for comparison. Note that we test fewer configurations for the latter index, as Retriever Top-k is fixed at a value of 50 for Azure AI Search.

Overall, we see that when using Elasticsearch, adding a rerank step improves performance for all three retrieval modes (sparse, dense, and hybrid), and that we achieve the highest performance on MRR@10 using hybrid retrieval (Retriever top-k=25) with a rerank step. We also note that on every configuration except dense retrieval without rerank, we achieve stronger MRR@10 performance using Elasticsearch as our index instead of Azure AI Search.

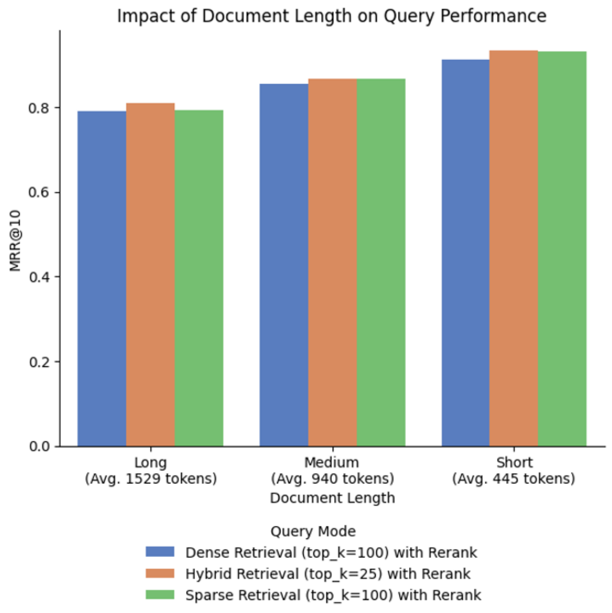

Fig. 4. Retrieval Performance by Document Length

With respect to retrieval time, queries run significantly faster on Azure AI Search than on our hosted Elasticsearch instance. This is largely due to differences in computing hardware between the two indexes, particularly for retrieval configurations involving vectorization and semantic rerank. While Azure AI Search vectorization and semantic rerank processes use Azure cloud GPU resources, the our hosted Elasticsearch instance runs these processes on a local PC without access to GPU processing. With proper GPU hardware, it is highly likely that Elasticsearch queries would be significantly faster than

Azure AI Search, given (1) the use of smaller models for vectorization and rerank, and (2) the lack of latency incurred in online API calls.

In addition to retrieval configurations, we also investigate how document length impacts query performance on average, as this is an additional factor that can be adjusted during document preprocessing using document chunking strategies. In Figure 4, we partition the CNN/Daily Mail into short, medium, and long document groups based on token length, then report MRR@10 within each group for the top 3 configurations identified in Table II. Here we observe that for each query configuration, MRR@10 increases substantially as average document length decreases. This suggests that the performance of our system could be improved by partitioning longer documents into smaller fragments.

B. Generation Experiments

We perform qualitative evaluation of each of our three candidate LLMs based on three specific criteria: (1) writing quality, (2) fidelity to the original retrieved text (i.e. groundedness), and (3) ability to follow different content and stylistic requirements (i.e. instruction following). Table III summarizes our findings, in which we score each model on a scale of 1-3 (3 being best) for each capability in a blind evaluation.

TABLE III
LLM Evaluation Summary

Model	Topic ID	Writing Quality	Groundedness	Instruction Following
Pegasus	1	2	3	1
Pegasus	2	3	2	1
Pegasus	3	2	3	1
Pegasus	4	3	3	1
Pegasus	5	2	3	1
Falcon	1	2	3	3
Falcon	2	2	3	2
Falcon	3	3	3	3
Falcon	4	2	3	3
Falcon	5	3	3	3
GPT-3.5-Turbo	1	3	3	3
GPT-3.5-Turbo	2	3	3	3
GPT-3.5-Turbo	3	2	3	2
GPT-3.5-Turbo	4	3	3	3
GPT-3.5-Turbo	5	3	3	3

Overall, we identify Falcon-RW-1B-Chat and GPT-3.5-Turbo as the only two viable candidate models based on performance on all three criteria. While all models return coherent, hallucination-free text for each topic, only the Falcon-RW-1B-Chat and GPT-3.5-Turbo models are capable of producing structurally diverse outputs to meet different section requirements. In contrast, the Pegasus model simply produces an abstractive summary for each section prompt, likely owing to its lack of explicit instruction tuning. To compare response quality between our two viable candidate models, we report outputs from both Falcon-RW-1B-Chat and GPT-3.5-Turbo for all three sections. For brevity, we limit outputs to those for Topic 1, though we observe similar results for the remaining four topics.

Outputs for Topic: Natural Disasters on the West Coast

Falcon-RW-1B-Chat

Generated Title: West Coast Braces for Winter Snowfall Caused By Huge Storms - Potential Flooding, Mudslides Ahead

Generated Highlights: 1. Winter storm with heavy snowfall and strong winds affecting the West Coast 2. Potential for mudslides and flooding due to rising sea levels 3. Flooding in the San Joaquin Valley, including Fresno and Sacramento

Conclusion: The devastating impact of natural disasters such as floods, landslides, and hurricanes continues to plague the West Coast, with many areas facing significant damage and disruption. As we have seen recently in the aftermath of Hurricane Irene and Tropical Storm Lee, these types of events can be extremely costly and require extensive resources to recover from. However, we must also recognize that these tragedies are preventable and that taking steps now to prepare and respond effectively can help minimize future losses. Ultimately, it is up to us to work together to ensure that we are prepared for the challenges ahead and that we prioritize sustainability and resilience at all times.

GPT-3.5-Turbo

Generated Title: West Coast Braces for Natural Disasters: Winter Storms, Floods, and Mudslides

Generated Highlights: 1. A winter storm is affecting the West Coast, causing flooding and potential mudslides in lower spots and impacting driving conditions and air travel. 2. Rising sea levels are elevating the flooding risk, and Americans need to stop behaving as though major disasters are rare and unpredictable. 3. Catastrophic disasters like Superstorm Sandy should be seen as teachable moments to better prepare the general public and elected officials for what is inevitable: the next storm.

Conclusion: In conclusion, natural disasters on the West Coast continue to pose a significant threat to both the environment and human life. With the increasing frequency and severity of these events, it is crucial that individuals and governments take proactive measures to mitigate their impact. This includes investing in infrastructure, emergency response, and public education to ensure that communities are prepared for the inevitable next storm. By working together and taking action now, we can help to minimize the devastating effects of natural disasters on the West Coast.

VI. Discussion

Results from both the retrieval and generation experiments demonstrate that our solution can effectively produce document sections using locally hosted information retrieval systems and either online, closed-source (Azure AI Search) or offline (Falcon-RW-1B-Chat) models.

With respect to retrieval, we show that, compared with the online Azure AI Search solution, Elasticsearch achieves slightly stronger performance on MRR@10 for most configurations tested, though more optimal hardware settings are necessary to achieve comparable query latency. We also note that the flexibility and interoperability of Elasticsearch is advantageous within the rapidly developing field of machine learning and natural language processing, as it allows us to more effectively adapt and improve our solution as new techniques and open-source models are released.

Several promising areas of exploration remain in terms of improving and expanding retrieval capabilities for our solution. With respect to retrieval methodologies, our hybrid retrieval approach can be extended with retriever ensembling, in which additional document scoring mechanisms (e.g. rankers that promote greater document diversity) can be combined with existing retrievers using RRF prior to semantic rerank. Other advanced RAG approaches, such as encoder model fine-tuning, sentence window retrieval or hypothetical document embeddings [25], are also viable approaches for continued exploration. Additionally, we can expand the application of our solution beyond purely text documents by incorporating vector-based image retrieval, employing multi-modal models such as CLIP [26] to encode text and images in the same vector space, enabling natural language image search.

We also show that, while GPT-3.5-Turbo is more reliable in terms of writing quality and instruction-following, comparable performance is achieved on shorter document sections using significantly smaller, open-source instruction-tuned chat models. Specifically, we demonstrate that the Falcon-RW-1B-Chat model is also capable of producing high-quality text based on a sizable (20 sentences) input context while meeting varying length and formatting requirements. Together, these online and offline model offerings serve different preferences with respect to cost, scale, and document sophistication. Specifically, GPT-3.5-Turbo incurs cost by usage and is better suited to smaller-scale (i.e. fewer users and lower document volume) use cases or those involving complex document formatting. In contrast, smaller, self-hosted offline models like Falcon-RW-1B-Chat incur fixed, recurring costs for compute in a typical production setting, and are better suited to larger-scale use cases or those with simpler document structure. We also note that self-hosted models in conjunction with Elasticsearch afford users the ability to keep data fully offline or within a controlled, private subnet, which is beneficial for clients with sensitive data or more restrictive policies regarding data transfer.

For large-scale use cases involving complex document structures, our solution can also be adapted to use significantly larger self-hosted open-source models (e.g. Falcon-40B and Llama-70B-Chat), however such models incur significantly higher cost due to their sizable compute requirements, and we leave their validation to future work. With respect to validating LLMs more broadly, an additional area of future work also includes developing a more systematic evaluation method for LLMs. This would increase the robustness of our performance measurements and our rate of iteration, allowing us to test more models, configurations, and prompts in a shorter amount of time. As LLM-based tools such as RAGAS [27] become increasingly reliable for long-form text evaluation, we plan to validate and adopt such tools for this purpose.

VII. Conclusion

Automated document composition is a nuanced and challenging endeavor, but one that enterprises stand to reap significant financial benefits from through increased employee productivity. Modern LLMs and developments in RAG research provide much of the tooling necessary to accomplish this, but challenges remain with respect to building scalable applications that are easy to use and adapt to different document types.

In this study, we present such an application and demonstrate that our approach is robust, performant, and adaptable to different use cases and cost and security preferences. Specifically, we explore the use of Elasticsearch, a secured database and search engine, against the popularly used Azure AI Search, and demonstrate through targeted measurement and optimization that the self-hosted Elasticsearch component can outperform enterprise-scale search engines. We also show through experimentation with the Falcon-RW-1B-Chat model that we can produce high-quality, formatted document text with relatively smaller, open-source generative models. Overall, this shows that it is possible to build an efficient, self-hosted solution that achieves comparable performance to solutions built using Azure OpenAI services while also providing greater data security, search engine customization, and cost flexibility.

Beyond RAG performance, we also showcase how specific software design patterns can enable our application to efficiently scale to arbitrary workloads and adapt to different use cases. Using Haystack or a similar modular pipeline builder allows developers to quickly update their application to test and adopt new open-source retriever and reranker models as they are released, accelerating the rate of application enhancement. Moreover, we further modularize our application by separating components of the RAG pipeline into distinct API services, enabling us to provide bespoke computing resources to each service and scale components independently of each other.

In summary, results from this study underscore how LLMs and retrieval-augmented generation, when coupled with intelligent application design, can provide enterprises with a flexible, robust, and user-friendly tool for automated document generation. This study aims to benefit researchers, data science practitioners, and software developers looking to build intelligent software to improve workforce productivity and improve the accessibility of generative AI technologies across their organizations.

VIII. Acknowledgments

This work is supported by the Accenture Federal Services (AFS) Discovery Lab, an organization that works with United States federal agencies to explore emerging technologies and prototype new capabilities in data science and artificial intelligence.

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021.

[2] H. Li, Y. Su, D. Cai, Y. Wang, and L. Liu, "A survey on retrieval-augmented text generation," 2022.

[3] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen, "Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy," 2023.

[4] B. Elasticsearch, "Elasticsearch," 2018.

[5] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 11 2019.

[6] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, Étienne Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, and G. Penedo, "The falcon series of open language models," 2023.

[7] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay, "The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only," 2023.

[8] Deepset-ai, "End-to-end python framework for building natural language search interfaces to data," 2021.

[9] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, "Realm: Retrieval-augmented language model pre-training," 2020.

[10] Z. Ji, Z. Lu, and H. Li, "An information retrieval approach to short text conversation," 2014.

[11] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. tau Yih, "Dense passage retrieval for open-domain question answering," 2020.

[12] Y. Zhang, S. Sun, X. Gao, Y. Fang, C. Brockett, M. Galley, J. Gao, and B. Dolan, "Retgen: A joint framework for retrieval and grounded text generation modeling," 2022.

[13] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," 2023.

[14] H.-T. Chen, F. Xu, S. Arora, and E. Choi, "Understanding retrieval augmentation for long-form question answering," 2023.

[15] H. Li, A. Einolghozati, S. Iyer, B. Paranjape, Y. Mehdad, S. Gupta, and M. Ghazvininejad, "Ease: Extractive-abstractive summarization with explanations," 2021.

[16] D. Miller, "Leveraging bert for extractive text summarization on lectures," 2019.

[17] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "Mpnet: Masked and permuted pre-training for language understanding," 2020.

[18] V. Gupta, M. Chinnakotla, and M. Shrivastava, "Retrieve and re-rank: A simple and effective IR approach to simple question answering over knowledge graphs," in *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)* (J. Thorne, A. Vlachos, O. Cocarascu, C. Christodoulopoulos, and A. Mittal, eds.), (Brussels, Belgium), pp. 22–27, Association for Computational Linguistics, Nov. 2018.

[19] G. Geigle, J. Pfeiffer, N. Reimers, I. Vulić, and I. Gurevych, "Retrieve fast, rerank smart: Cooperative and joint approaches for improved cross-modal retrieval," 2022.

[20] M. Glass, G. Rossiello, M. F. M. Chowdhury, A. R. Naik, P. Cai, and A. Gliozzo, "Re2g: Retrieve, rerank, generate," 2022.

[21] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, dense, and attentional representations for text retrieval," 2021.

[22] G. Amati, "Bm25," 2009.

[23] G. V. Cormack, C. L. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," 2009.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[25] L. Gao, X. Ma, J. Lin, and J. Callan, "Precise zero-shot dense retrieval without relevance labels," 2022.

[26] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021.

[27] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated Evaluation of Retrieval Augmented Generation," 2023.