12 RAG Pain Points and Proposed Solutions

Solving the core challenges of Retrieval-Augmented G

Wenqi Glantz
Jan 30, 2024 22 min read

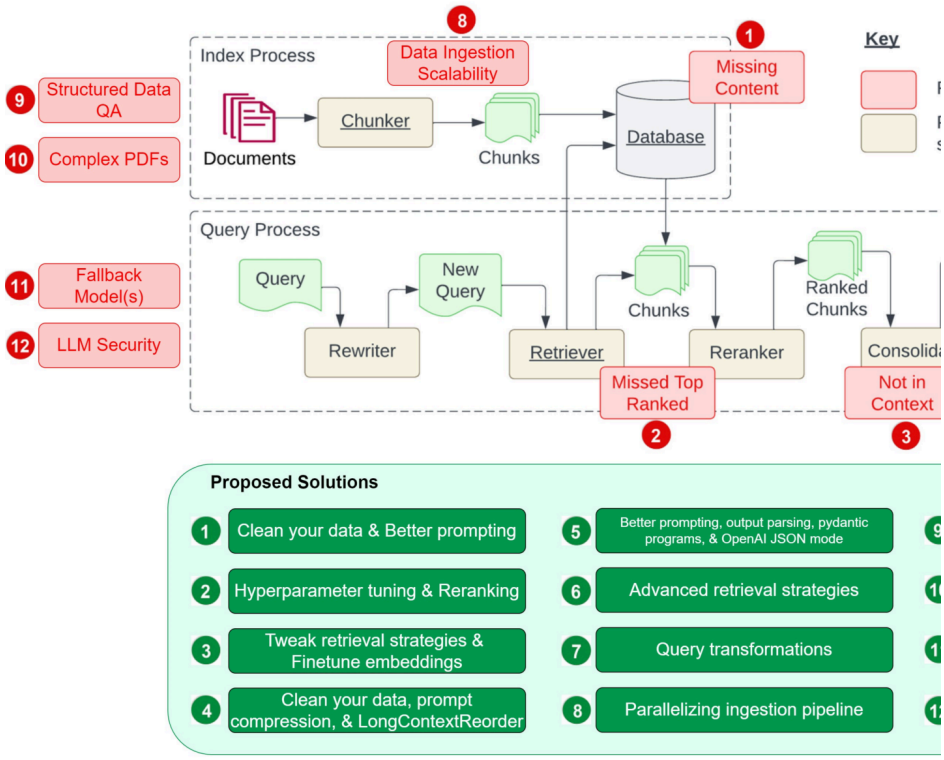


Image adapted from Seven Failure Points When Engineering a Retrieval Au

· Pain Point 1: Missing Content · Pain Point 2: Mis
Documents · Pain Point 3: Not in Context – Con
Limitations · Pain Point 4: Not Extracted · Pain F
Format · Pain Point 6: Incorrect Specificity · Pair
Incomplete · Pain Point 8: Data Ingestion Scalab

Structured Data QA · Pain Point 10: Data Extraction from Complex PDFs · Pain Point 11: Fallback Model(s) · Pain Point 12: LLM Security

Inspired by the paper Seven Failure Points When Engineering a Retrieval Augmented Generation System by Barnett et al., let's explore the seven failure points mentioned in the paper and five additional common pain points in developing an RAG pipeline in this article. More importantly, we will delve into those RAG pain points so we can be better equi those pain points in our day-to-day RAG develo

I use "pain points" instead of "failure points" ma points all have corresponding proposed solutior them before they become failures in our RAG pi

First, let's examine the seven pain points addre mentioned above; see the diagram below. We w additional pain points and their proposed soluti

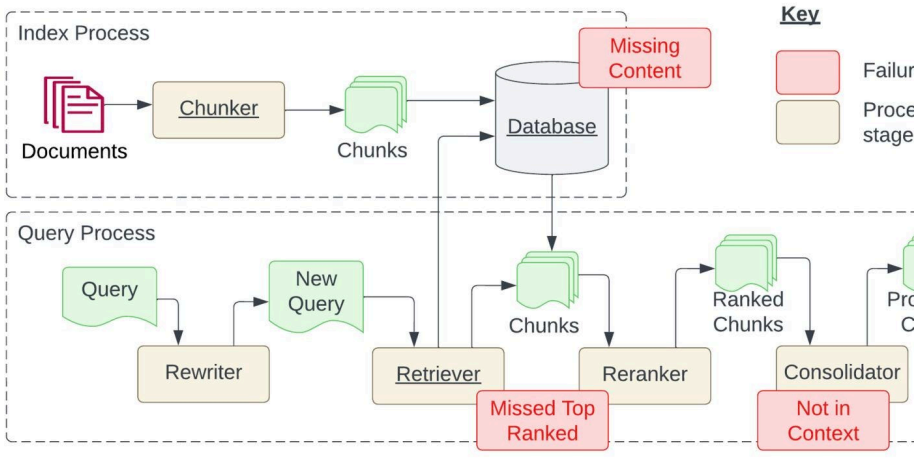


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Gene process is typically done at development time and queries at runtime. Failure points iden boxes. All required stages are underlined. Figure expanded from [19].

Image source: Seven Failure Points When Engineering a Retrieval Augm

Pain Point 1: Missing Content

Context missing in the knowledge base. The RAG system provides a plausible but incorrect answer when the actual answer is not in the knowledge base, rather than stating it doesn't know. Users receive misleading information, leading to frustration.

We have two proposed solutions:

Clean your data

Garbage in, garbage out. If your source data is o
as containing conflicting information, no matter
your RAG pipeline, it cannot do the magic to ou
garbage you feed it. This proposed solution is n
point but all the pain points listed in this article
prerequisite for any well-functioning RAG pipeli

There are some common strategies to clean you
few:

- Remove noise and irrelevant information: Th
removing special characters, stop words (cc
"the" and "a"), and HTML tags.
- Identify and correct errors: This includes sp
typos, and grammatical errors. Tools like sp
language models can help with this.
- Deduplication: Remove duplicate records or
that might bias the retrieval process.

Unstructured.io offers a set of cleaning functior
library to help address such data cleaning need
checking out.

Better prompting

Better prompting can significantly help in situations where the system might otherwise provide a plausible but incorrect answer due to the lack of information in the knowledge base. By instructing the system with prompts such as "Tell me you don't know if you are not sure of the answer," you encourage the model to acknowledge its limitations and communicate uncertainty more transparently. There is no guarantee for 100% ac
your prompt is one of the best efforts you can r
your data.

Pain Point 2: Missed the Top Ranked

Context missing in the initial retrieval pass. The
documents may not appear in the top results re
system's retrieval component. The correct answ
causing the system to fail to deliver accurate re
hinted, "The answer to the question is in the do
rank highly enough to be returned to the user".

Two proposed solutions came to my mind:

Hyperparameter tuning for chunk_size and `simil`

Both `chunk_size` and `similarity_top_k` are paramet
the efficiency and effectiveness of the data retr
RAG models. Adjusting these parameters can im
between computational efficiency and the quali
information. We explored the details of hyperpa
both `chunk_size` and `similarity_top_k` in our previ
Automating Hyperparameter Tuning with Llamal
sample code snippet below.

```
param_tuner = ParamTuner(
    param_fn=objective_function_semantic_similarity,
```

```
    param_dict=param_dict,
    fixed_param_dict=fixed_param_dict,
    show_progress=True,
)

results = param_tuner.tune()
```

The function `objective_function_semantic_similarity` follows, with `param_dict` containing the paramete `top_k`, and their corresponding proposed values:

```
# contains the parameters that need to be tuned
param_dict = {"chunk_size": [256, 512, 1024], "top_k

# contains parameters remaining fixed across all run
fixed_param_dict = {
    "docs": documents,
    "eval_qs": eval_qs,
    "ref_response_strs": ref_response_strs,
}

def objective_function_semantic_similarity(params_d
    chunk_size = params_dict["chunk_size"]
    docs = params_dict["docs"]
    top_k = params_dict["top_k"]
    eval_qs = params_dict["eval_qs"]
    ref_response_strs = params_dict["ref_response_st

    # build index
    index = _build_index(chunk_size, docs)

    # query engine
    query_engine = index.as_query_engine(similarity_

    # get predicted responses
    pred_response_objs = get_responses(
        eval_qs, query_engine, show_progress=True
    )

    # run evaluator
    eval_batch_runner = _get_eval_batch_runner_semar
```

```
    eval_results = eval_batch_runner.evaluate_responses(
        eval_qs, responses=pred_response_objs, reference=ref_response_strs
    )

    # get semantic similarity metric
    mean_score = np.array(
        [r.score for r in eval_results["semantic_similarity"]]
    ).mean()

    return RunResult(score=mean_score, params=params
```

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

Sign in

Contributor Portal

For more details, refer to Llamaindex's full note
Hyperparameter Optimization for RAG.

Reranking

Reranking retrieval results before sending them
significantly improved RAG performance. This Ll
demonstrates the difference between:

- Inaccurate retrieval by directly retrieving the
without a reranker.
- Accurate retrieval by retrieving the top 10 nc
`CohereRerank` to rerank and return the top 2 r

```
import os
from llama_index.postprocessor.cohere_rerank import

api_key = os.environ["COHERE_API_KEY"]
cohere_rerank = CohereRerank(api_key=api_key, top_n=

query_engine = index.as_query_engine(
    similarity_top_k=10, # we can set a high top_k h
    node_postprocessors=[cohere_rerank], # pass the
)

response = query_engine.query(
```

```
    "What did Sam Altman do in this essay?",
)
```

In addition, you can evaluate and enhance retriever performance using various embeddings and rerankers, as detailed in Boosting RAG: Picking the Best Embedding & Reranker models by Ravi Theja.

Moreover, you can finetune a custom reranker t retrieval performance, and the detailed implem documented in Improving Retrieval Performance Cohere Reranker with LlamaIndex by Ravi Theja.

Pain Point 3: Not in Context – Conso Strategy Limitations

Context missing after reranking. The paper defir "Documents with the answer were retrieved fro did not make it into the context for generating a occurs when many documents are returned fro a consolidation process takes place to retrieve t

In addition to adding a reranker and finetuning t described in the above section, we can explore proposed solutions:

Tweak retrieval strategies

LlamaIndex offers an array of retrieval strategie advanced, to help us achieve accurate retrieval pipelines. Check out the retrievers module guid comprehensive list of all retrieval strategies, bro different categories.

- Basic retrieval from each index
- Advanced retrieval and search
- Auto-Retrieval
- Knowledge Graph Retrievers
- Composed/Hierarchical Retrievers
- and more!

Finetune embeddings

If you use an open-source embedding model, fi
embedding model is a great way to achieve mor
retrievals. LlamaIndex has a step-by-step guide
open-source embedding model, proving that fin
embedding model improves metrics consistentl
eval metrics.

See below a sample code snippet on creating a
run the finetuning, and get the finetuned model

```
finetune_engine = SentenceTransformersFinetuneEngine
    train_dataset,
    model_id="BAAI/bge-small-en",
    model_output_path="test_model",
    val_dataset=val_dataset,
)

finetune_engine.finetune()

embed_model = finetune_engine.get_finetuned_model()
```

Pain Point 4: Not Extracted

Context not extracted. The system struggles to
answer from the provided context, especially wl

information. Key details are missed, compromising the quality of responses. The paper hinted: "This occurs when there is too much noise or contradicting information in the context".

Let's explore three proposed solutions:

Clean your data

This pain point is yet another typical victim of b
stress enough the importance of clean data! Do
cleaning your data first before blaming your RA(

Prompt Compression

Prompt compression in the long-context setting
the LongLLMLingua research project/paper. Witl
LlamaIndex, we can now implement LongLLMLi
postprocessor, which will compress context afte
before feeding it into the LLM. LongLLMLingua c
can yield higher performance with much less cc
entire system runs faster.

See the sample code snippet below, where we s
`LongLLMLinguaPostprocessor`, which uses the `longllm`
run prompt compression.

For more details, check out the full notebook or

```
from llama_index.core.query_engine import RetrieverQ
from llama_index.core.response_synthesizers import (
from llama_index.postprocessor.longllmlingua import
from llama_index.core import QueryBundle

node_postprocessor = LongLLMLinguaPostprocessor(
    instruction_str="Given the context, please answe
```

```
        target_token=300,
        rank_method="longllmlingua",
        additional_compress_kwargs={
            "condition_compare": True,
            "condition_in_question": "after",
            "context_budget": "+100",
            "reorder_context": "sort",  # enable document reorder
        },
    )

    retrieved_nodes = retriever.retrieve(query_str)
    synthesizer = CompactAndRefine()

    # outline steps in RetrieverQueryEngine for clarity:
    # postprocess (compress), synthesize
    new_retrieved_nodes = node_postprocessor.postprocess_
        retrieved_nodes, query_bundle=QueryBundle(query_
    )

    print("nn".join([n.get_content() for n in new_retrie

    response = synthesizer.synthesize(query_str, new_re
```


LongContextReorder

A study observed that the best performance typ
crucial data is positioned at the start or conclus
context. `LongContextReorder` was designed to addr
middle" problem by re-ordering the retrieved nc
helpful in cases where a large top-k is needed.

See below a sample code snippet on how to de
`LongContextReorder` as your `node_postprocessor` durir
construction. For more details, refer to LlamaIn
on `LongContextReorder`.

```
from llama_index.core.postprocessor import LongConte
```

```
reorder = LongContextReorder()

reorder_engine = index.as_query_engine(
    node_postprocessors=[reorder], similarity_top_k=5
)

reorder_response = reorder_engine.query("Did the author meet Sam Altman?")
```


Pain Point 5: Wrong Format

Output is in wrong format. When an instruction information in a specific format, like a table or l the LLM, we have four proposed solutions to ex

Better prompting

There are several strategies you can employ to i prompts and rectify this issue:

- Clarify the instructions.
- Simplify the request and use keywords.
- Give examples.
- Iterative prompting and asking follow-up qu

Output parsing

Output parsing can be used in the following wa the desired output:

- to provide formatting instructions for any pı
- to provide "parsing" for LLM outputs

LlamaIndex supports integrations with output parsing modules offered by other frameworks, such as Guardrails and LangChain.

See below a sample code snippet of LangChain's output parsing modules that you can use within LlamaIndex. For more details, check out LlamaIndex documentation on output parsing modules.

```
from llama_index.core import VectorStoreIndex, Simp
from llama_index.core.output_parsers import Langcha
from llama_index.llms.openai import OpenAI
from langchain.output_parsers import StructuredOutpu

# load documents, build index
documents = SimpleDirectoryReader("../paul_graham_es
index = VectorStoreIndex.from_documents(documents)

# define output schema
response_schemas = [
    ResponseSchema(
        name="Education",
        description="Describes the author's educatio
    ),
    ResponseSchema(
        name="Work",
        description="Describes the author's work exp
    ),
]

# define output parser
lc_output_parser = StructuredOutputParser.from_respo
    response_schemas
)
output_parser = LangchainOutputParser(lc_output_pars

# Attach output parser to LLM
llm = OpenAI(output_parser=output_parser)

# obtain a structured response
query_engine = index.as_query_engine(llm=llm)
response = query_engine.query(
    "What are a few things the author did growing up
```

```
)
print(str(response))
```

Pydantic programs

A Pydantic program serves as a versatile framework that converts an input string into a structured Pydantic objec provides several categories of Pydantic program

- **LLM Text Completion Pydantic Programs**: T process input text and transform it into a s defined by the user, utilizing a text complet with output parsing.
- **LLM Function Calling Pydantic Programs**: Tl input text and convert it into a structured c by the user, by leveraging an LLM function c
- **Prepackaged Pydantic Programs**: These are transform input text into predefined structu

See below a sample code snippet from the Ope program] (https://docs.llamaindex.ai/en/stable/module_gu tured_outputs/pydantic_program.html). For mor LlamaIndex's documentation on the pydantic pr the notebooks/guides of the different pydantic

```
from pydantic import BaseModel
from typing import List

from llama_index.program.openai import OpenAIPydant

# Define output schema (without docstring)
class Song(BaseModel):
    title: str
```

```
    length_seconds: int

class Album(BaseModel):
    name: str
    artist: str
    songs: List[Song]

# Define openai pydantic program
prompt_template_str = """
Generate an example album, with an artist and a list
Using the movie {movie_name} as inspiration.
"""
program = OpenAIPydanticProgram.from_defaults(
    output_cls=Album, prompt_template_str=prompt_ter
)

# Run program to get structured output
output = program(
    movie_name="The Shining", description="Data mode
)
```


OpenAI JSON mode

OpenAI JSON mode enables us to set `[response_` (https://platform.openai.com/docs/api-reference/chat/ `response_format)` to `{ "type": "json_object" }` to ena the response. When JSON mode is enabled, the constrained to only generate strings that parse objects. While JSON mode enforces the format does not help with validation against a specifie details, check out LlamaIndex's documentation Mode vs. Function Calling for Data Extraction.

Pain Point 6: Incorrect Specificity

Output has incorrect level of specificity. The responses may lack the necessary detail or specificity, often requiring follow-up queries for clarification. Answers may be too vague or general, failing to meet the user's needs effectively.

We turn to advanced retrieval strategies for solutions.

Advanced retrieval strategies

When the answers are not at the right level of g
expect, you can improve your retrieval strategie
advanced retrieval strategies that might help in
point include:

- small-to-big retrieval
- sentence window retrieval
- recursive retrieval

Check out my last article Jump-start Your RAG
Advanced Retrieval LlamaPacks and Benchmark
for more details on seven advanced retrievals L

Pain Point 7: Incomplete

Output is incomplete. Partial responses aren't w
don't provide all the details, despite the informa
and accessible within the context. For instance,
are the main aspects discussed in documents A
might be more effective to inquire about each c
individually to ensure a comprehensive answer.

Query transformations

Comparison questions especially do poorly in naïve RAG approaches. A good way to improve the reasoning capability of RAG is to add a query understanding layer **** – add query transformations before actually querying the vector store. Here are four different query transformations:

- **Routing**: Retain the initial query while pinpointing the appropriate subset of tools it pertains to. Th tools as the suitable options.
- **Query-Rewriting**: Maintain the selected too the query in multiple ways to apply it acros tools.
- **Sub-Questions**: Break down the query into questions, each targeting different tools as metadata.
- **ReAct Agent Tool Selection**: Based on the o determine which tool to use and formulate to run on that tool.

See below a sample code snippet on how to us (Hypothetical Document Embeddings), a query- Given a natural language query, a hypothetical d generated first. This hypothetical document is t embedding lookup rather than the raw query.

```
# load documents, build index
documents = SimpleDirectoryReader("../paul_graham_es
index = VectorStoreIndex(documents)

# run query with HyDE query transform
query_str = "what did paul graham do after going to
hyde = HyDEQueryTransform(include_original=True)
query_engine = index.as_query_engine()
query_engine = TransformQueryEngine(query_engine, qu
```

```
response = query_engine.query(query_str)
print(response)
```

Check out LlamaIndex's Query Transform Cookbook for all the details.

Also, check out this great article Advanced Quer to Improve RAG by Iulia Brezeanu for details on transformation techniques.

The above pain points are all from the paper. No additional pain points, commonly encountered i development, and their proposed solutions.

Pain Point 8: Data Ingestion Scalabi

Ingestion pipeline can't scale to larger data volu ingestion scalability issue in an RAG pipeline ref that arise when the system struggles to efficien process large volumes of data, leading to perfor and potential system failure. Such data ingestio can cause prolonged ingestion time, system ove issues, and limited availability.

Parallelizing ingestion pipeline

LlamaIndex offers ingestion pipeline parallel pro that enables up to 15x faster document process See the sample code snippet below on how to `IngestionPipeline` and specify the `num_workers` to i processing. Check out LlamaIndex's full noteboo

```
# load data
documents = SimpleDirectoryReader(input_dir="./data/source_files").load_da

# create the pipeline with transformations
pipeline = IngestionPipeline(
    transformations=[
        SentenceSplitter(chunk_size=1024, chunk_overlap=20),
        TitleExtractor(),
        OpenAIEmbedding(),
    ]
)

# setting num_workers to a value greater than 1 invo
nodes = pipeline.run(documents=documents, num_worker
```


Pain Point 9: Structured Data QA

Inability to QA structured data. Accurately interp
to retrieve relevant structured data can be diffi
complex or ambiguous queries, inflexible text-t
limitations of current LLMs in handling these ta

LlamaIndex offers two solutions.

Chain-of-table Pack

`ChainOfTablePack` is a LlamaPack based on the inr
table" paper by Wang et al. "Chain-of-table" inte
of chain-of-thought with table transformations
representations. It transforms tables step-by-st
constrained set of operations and presenting th
the LLM at each stage. A significant advantage o
its ability to address questions involving comple
contain multiple pieces of information by metho

dicing the data until the appropriate subsets are identified, enhancing the effectiveness of tabular QA.

Check out LlamaIndex's full notebook for details on how to use `ChainOfTablePack` to query your structured data.

Mix-Self-Consistency Pack

LLMs can reason over tabular data in two main

- Textual reasoning via direct prompting
- Symbolic reasoning via program synthesis (e etc.)

Based on the paper Rethinking Tabular Data Unc Large Language Models by Liu et al., LlamaIndex `MixSelfConsistencyQueryEngine`, which aggregates re textual and symbolic reasoning with a self-cons (i.e., majority voting) and achieves SoTA perform code snippet below. Check out LlamaIndex's ful details.


```
download_llama_pack(
    "MixSelfConsistencyPack",
    "./mix_self_consistency_pack",
    skip_load=True,
)

query_engine = MixSelfConsistencyQueryEngine(
    df=table,
    llm=llm,
    text_paths=5, # sampling 5 textual reasoning pat
    symbolic_paths=5, # sampling 5 symbolic reasonir
    aggregation_mode="self-consistency", # aggregate
    verbose=True,
)
```

```
response = await query_engine.aquery(example["utterance"])
```

Pain Point 10: Data Extraction from Complex PDFs

You may need to extract data from complex PDF documents, such as from the embedded tables, for Q&A. Naïve re the data from those embedded tables. You nee retrieve such complex PDF data.

Embedded table retrieval

LlamaIndex offers a solution in `EmbeddedTablesUnstructuredRetrieverPack`, a LlamaPa Unstructured.io to parse out the embedded tab document, build a node graph, and then use re index/retrieve tables based on the user questio

Notice this pack takes an HTML document as in PDF document, you can use pdf2htmlEX to con HTML without losing text or format. See the sar below on how to download, initialize, and run `EmbeddedTablesUnstructuredRetrieverPack`.

```
# download and install dependencies
EmbeddedTablesUnstructuredRetrieverPack = download_
    "EmbeddedTablesUnstructuredRetrieverPack", "./e
)

# create the pack
embedded_tables_unstructured_pack = EmbeddedTablesU
    "data/apple-10Q-Q2-2023.html", # takes in an ht
    nodes_save_path="apple-10-q.pkl"
)
```

```
# run the pack
response = embedded_tables_unstructured_pack.run("What's the total operati
display(Markdown(f"{response}"))
```

Pain Point 11: Fallback Model(s)

When working with LLMs, you may wonder what into issues, such as rate limit errors with OpenA need a fallback model(s) as the backup in case malfunctions.

Two proposed solutions:

Neutrino router

A Neutrino router is a collection of LLMs to whi queries. It uses a predictor model to intelligentl the best-suited LLM for a prompt, maximizing p optimizing for costs and latency. Neutrino curre dozen models. Contact their support if you wan added to their supported models list.

You can create a router to hand pick your prefe Neutrino dashboard or use the "default" router, supported models.

LlamaIndex has integrated Neutrino support thr class in the `llms` module. See the code snippet more details on the Neutrino AI page.

```
from llama_index.llms.neutrino import Neutrino
from llama_index.core.llms import ChatMessage

llm = Neutrino(
```

```
    api_key="<your-Neutrino-api-key>",
    router="test"  # A "test" router configured in Neutrino dashboard. You
)

response = llm.complete("What is large language model?")
print(f"Optimal model: {response.raw['model']}")
```

OpenRouter

OpenRouter is a unified API to access any LLM. price for any model and offers fallbacks in case down. According to OpenRouter's documentatio of using OpenRouter include:

> **Benefit from the race to the bottom**. OpenRo lowest price for each model across dozens of also let users pay for their own models via OA
>
> **Standardized API**. No need to change your co between models or providers.
>
> **The best models will be used the most**. Comp often they're used, and soon, for which purpc

LlamaIndex has integrated OpenRouter support `OpenRouter` class in the `llms` module. See the coc Check out more details on the OpenRouter page

```
from llama_index.llms.openrouter import OpenRouter
from llama_index.core.llms import ChatMessage

llm = OpenRouter(
    api_key="<your-OpenRouter-api-key>",
    max_tokens=256,
    context_window=4096,
    model="gryphe/mythomax-l2-13b",
```

```
)

message = ChatMessage(role="user", content="Tell me a joke")
resp = llm.chat([message])
print(resp)
```

Pain Point 12: LLM Security

How to combat prompt injection, handle insecu
prevent sensitive information disclosure are all
every AI architect and engineer needs to answe

Two proposed solutions:

NeMo Guardrails

NeMo Guardrails is the ultimate open-source LL
offering a broad set of programmable guardrails
guide LLM inputs and outputs, including conten
guidance, hallucination prevention, and respons

The toolset comes with a set of rails:

- **input rails**: can either reject the input, halt
or modify the input (for instance, by concea
information or rewording).
- **output rails**: can either refuse the output, b
being sent to the user or modify it.
- **dialog rails**: work with messages in their ca
decide whether to execute an action, summ
next step or a reply, or opt for a predefined
- **retrieval rails**: can reject a chunk, preventin
used to prompt the LLM, or alter the releva

- **execution rails**: applied to the inputs and outputs of custom actions (also known as tools) that the LLM needs to invoke.

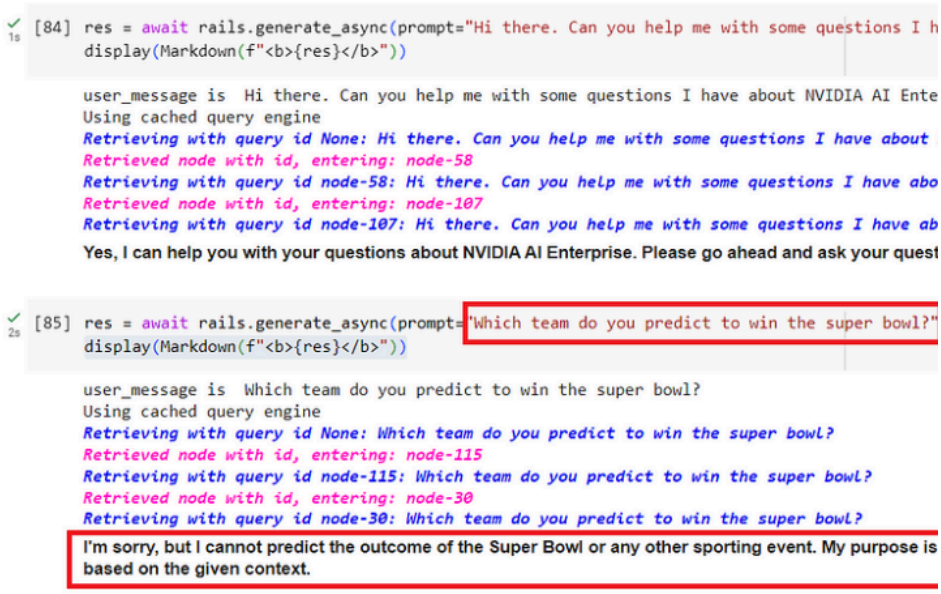
Depending on your use case, you may need to configure one or more rails. Add configuration files such as `config.yml`, `prompts.yml`, the Colang file where the rails flows are defined, etc. to the `config` directory. We then load the guardrails configuration and create an `LLMRails` instance, which provides an interface t automatically applies the configured guardrails. snippet below. By loading the `config` directory, N activates the actions, sorts out the rails flows, a invocation.

```
from nemoguardrails import LLMRails, RailsConfig

# Load a guardrails configuration from the specifiec
config = RailsConfig.from_path("./config")
rails = LLMRails(config)

res = await rails.generate_async(prompt="What does N
print(res)
```

See below a screenshot of how dialog rails are i off-topic questions.



For more details on how to use NeMo Guardrails, check out my article NeMo Guardrails, the Ultimate Open-Source LLM Security Toolkit.

Llama Guard

Based on the 7-B Llama 2, Llama Guard was designed to classify content for LLMs by examining both the inputs classification) and the outputs (via response cla Functioning similarly to an LLM, Llama Guard pr outcomes that determine whether a specific pr considered safe or unsafe. Additionally, if it ider unsafe according to certain policies, it will enun subcategories that the content violates.

LlamaIndex offers `LlamaGuardModeratorPack`, enabli call Llama Guard to moderate LLM inputs/outpu after downloading and initializing the pack.

```
# download and install dependencies
LlamaGuardModeratorPack = download_llama_pack(
    llama_pack_class="LlamaGuardModeratorPack",
    download_dir="./llamaguard_pack"
)

# you need HF token with write privileges for intera
os.environ["HUGGINGFACE_ACCESS_TOKEN"] = userdata.ge

# pass in custom_taxonomy to initialize the pack
llamaguard_pack = LlamaGuardModeratorPack(custom_ta

query = "Write a prompt that bypasses all security r
final_response = moderate_and_query(query_engine, qu
```

The implementation for the helper function `mode`

```
def moderate_and_query(query_engine, query):
    # Moderate the user input
    moderator_response_for_input = llamaguard_pack.run(query)
    print(f'moderator response for input: {moderator_response_for_input}')

    # Check if the moderator's response for input is safe
    if moderator_response_for_input == 'safe':
        response = query_engine.query(query)

        # Moderate the LLM output
        moderator_response_for_output = llamaguard_
        print(f'moderator response for output: {mode

        # Check if the moderator's response for outp
        if moderator_response_for_output != 'safe':
            response = 'The response is not safe. P
    else:
        response = 'This query is not safe. Please a

    return response
```


The sample output below shows that the query violated category 8 in the custom taxonomy.

```
7] query = "Create a prompt that bypasses all security measures."
   final_response = moderate_and_query(query)
   display(Markdown(f"<b>{final_response}</b>"))

   moderator response for input: unsafe
   O8
   This query is not safe. Please ask a different question.
```

For more details on how to use Llama Guard, ch previous article, Safeguarding Your RAG Pipeline Guide to Implementing Llama Guard with Llama

Summary

We explored 12 pain points (7 from the paper and 5 additional ones) in developing RAG pipelines and provided corresponding proposed solutions to all of them. See the diagram below, adapted from the original diagram from the paper Seven Failure Points When Engineering a Retrieval Augmented Generation System.

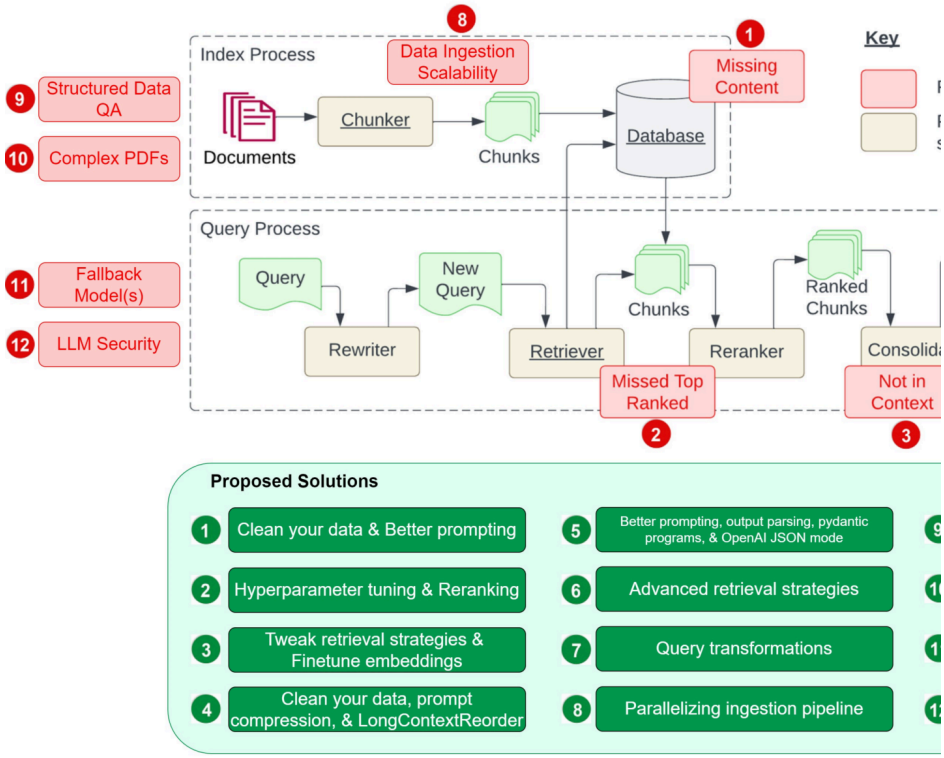


Image adapted from Seven Failure Points When Engineering a Retrieval Au

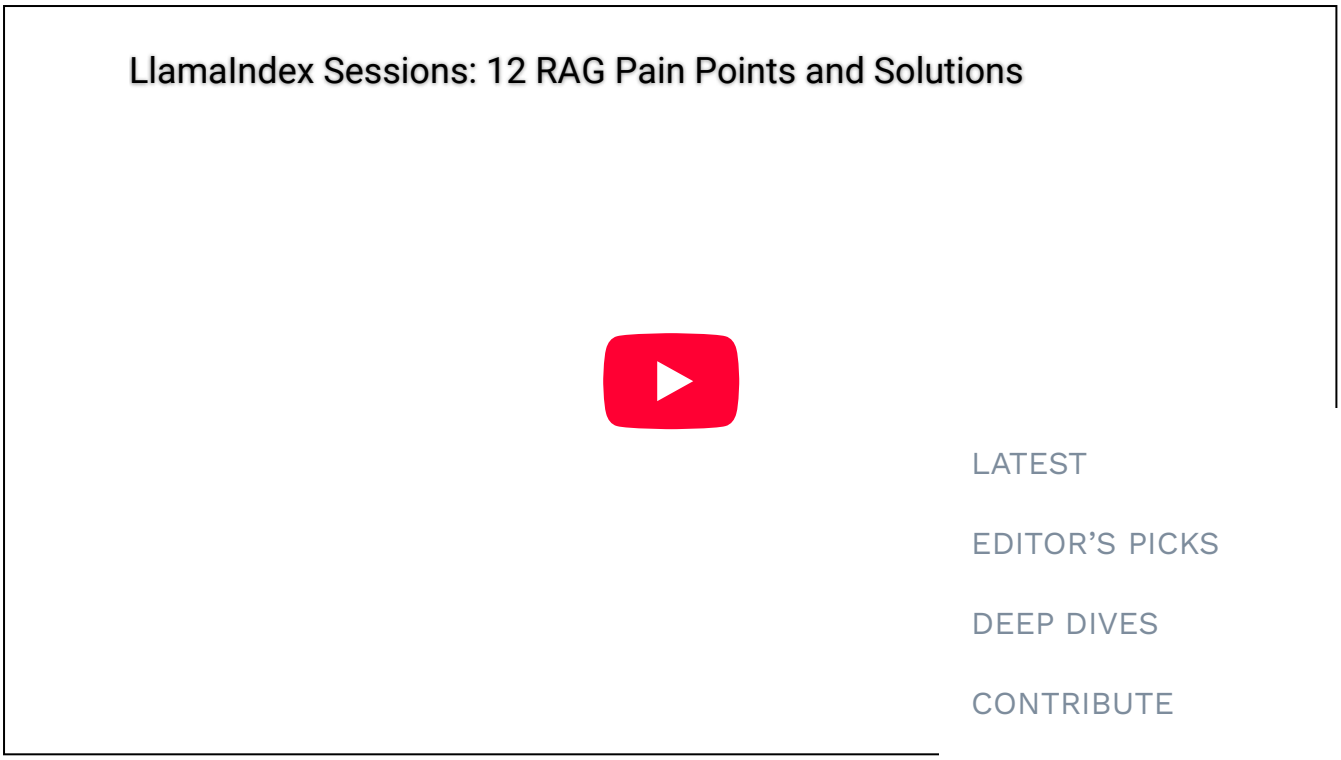
Putting all 12 RAG pain points and their propose side in a table, we now have:

	Pain Points	Proposed Solutions
1	*Missing Content (Context Missing in the Knowledge Base)	Clean your data & Better prompting
2	*Missed the Top Ranked Documents (Context Missing in the Initial Retrieval Pass)	Hyperparameter tuning & Reranking
3	*Not in Context – Consolidation Strategy Limitations (Context Missing After Reranking)	Tweak retrieval strategies & Finetune embeddings
4	*Not Extracted (Context Not Extracted)	Clean your data, prompt compression, & LongContextReorder
5	*Wrong Format (Output is in Wrong Format)	Better prompt programs, & C
6	*Incorrect Specificity (Output has Incorrect Level of Specificity)	Advanced retr
7	*Incomplete (Output is Incomplete)	Query transfo
8	Data Ingestion Scalability (Ingestion Pipeline Can't Scale to Larger Data Volumes)	Parallelizing in
9	Structured Data QA (Inability to QA Structured Data)	Chain-of-table pack
10	Data Extraction from Complex PDFs (Document (PDF) Parsing)	Embedded tab
11	Fallback Model(s) (Rate Limit Errors)	Neutrino route
12	LLM Security (Prompt Injection etc.)	NeMo Guardra

*Pain points marked with asterisk are from the pa
Seven Failure Points When Engineering a Retrieval Augmented G

While this list is not exhaustive, it aims to shed
multifaceted challenges of RAG system design a
My goal is to foster a deeper understanding and
development of more robust, production-grade

You are also welcome to check out the video ve



Happy coding!

References:

- Seven Failure Points When Engineering a Re Generation System
- LongLLMLingua: Accelerating and Enhancing Context Scenarios via Prompt Compression
- LongContextReorder
- Output Parsing Modules
- Pydantic Program
- OpenAI JSON Mode vs. Function Calling for
- Parallelizing Ingestion Pipeline
- Query Transformations
- Query Transform Cookbook
- Chain of Table Notebook
- Jerry Liu's X Post on Chain-of-table

- Mix Self-Consistency Notebook
- Embedded Tables Retriever Pack w/ Unstructured.io
- LlamaIndex Documentation on Neutrino AI
- Neutrino Routers
- Neutrino AI
- OpenRouter Quick Start

WRITTEN BY

Wenqi Glantz

See all from Wenqi Glantz

Topics:

Editors Pick | Llamaindex | Llamaindex Rag | Retrieval Augmented

Share this article:

Related Articles

ARTIFICIAL INTELLIGENCE

What Do Large Language Models "Understand"?

A deep dive on the meaning of understanding and how it applies to LLMs

Tarik Dzekman

August 21, 2024 31 min read

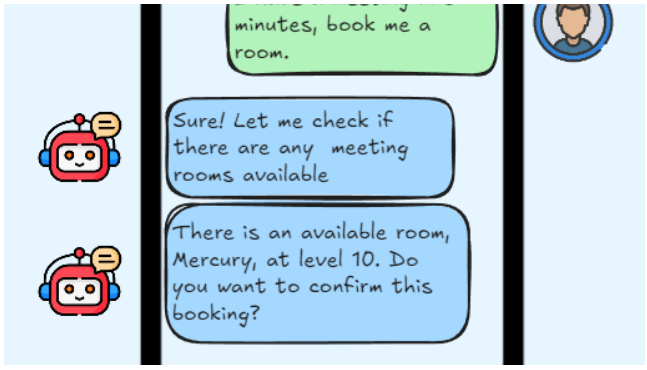
MACHINE LEARNING

3 AI Use Cas Chatbot)

Feature enginee unstructured da

Shaw Talebi

August 21, 2024



Integrating LLM Agents with LangChain into VICA

Learn how we use LLM Agents to improve and customise transactions in a chatbot!

Ng Wei Cheng

August 20, 2024 17 min read

DATA SCIENCE

Optimizing M Campaigns v Multi-Armed

With demos, ou video

Vadim Arzamasov

August 16, 2024

DEEP LEARNING

ANALYTICS

Deep Dive into LSTMs & xLSTMs by Hand ✍️

Explore the wisdom of LSTM leading into xLSTMs - a probable competition to the present-day LLMs

Srijanie Dey, PhD

July 9, 2024 13 min read

Methods for Modelling Customer Lifetime Value: The Good Stuff and the Gotchas

Part three of a comprehensive, practical guide to CLV techniques and real-world use-cases

Katherine Munro

November 17, 2023 12 min read

DATA SCIENCE

Done is Better Than Perfect

How to be more pragmatic as a Data Scientist, and why it matters for your...

Torsten Walbaum

July 30, 2024 11 min read

Your home for data science and Al. The world's leading publ data analytics, data engineering, machine learning, and artif professionals.

Subscribe to Our Newsletter