12 RAG Pain Points and Proposed Solutions

Solving the core challenges of Retrieval-Augmented Ge

Wenqi Glantz
Jan 30, 2024 22 min read

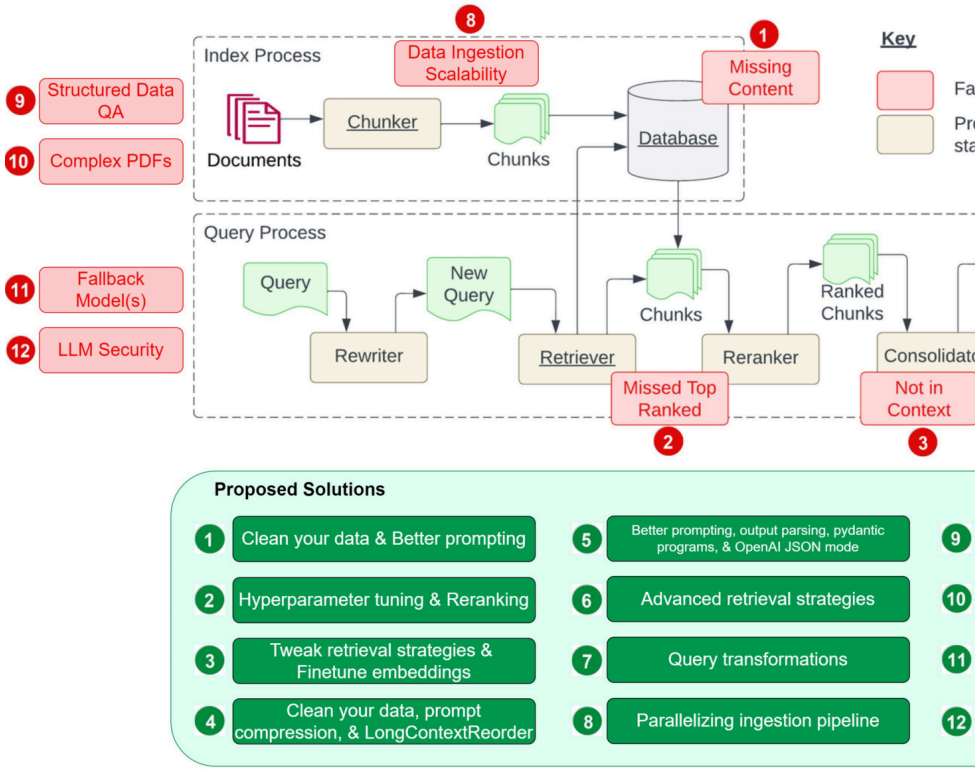


Image adapted from Seven Failure Points When Engineering a Retrieval Augr

· Pain Point 1: Missing Content · Pain Point 2: Miss
Documents · Pain Point 3: Not in Context – Consc
Limitations · Pain Point 4: Not Extracted · Pain Po
· Pain Point 6: Incorrect Specificity · Pain Point 7:

Point 8: Data Ingestion Scalability · Pain Point 9: 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 F article. More importantly, we will delve into the so RAG pain points so we can be better equipped to points in our day-to-day RAG development.

I use "pain points" instead of "failure points" mair points all have corresponding proposed solutions. them before they become failures in our RAG pipe

First, let's examine the seven pain points address mentioned above; see the diagram below. We will additional pain points and their proposed solutior

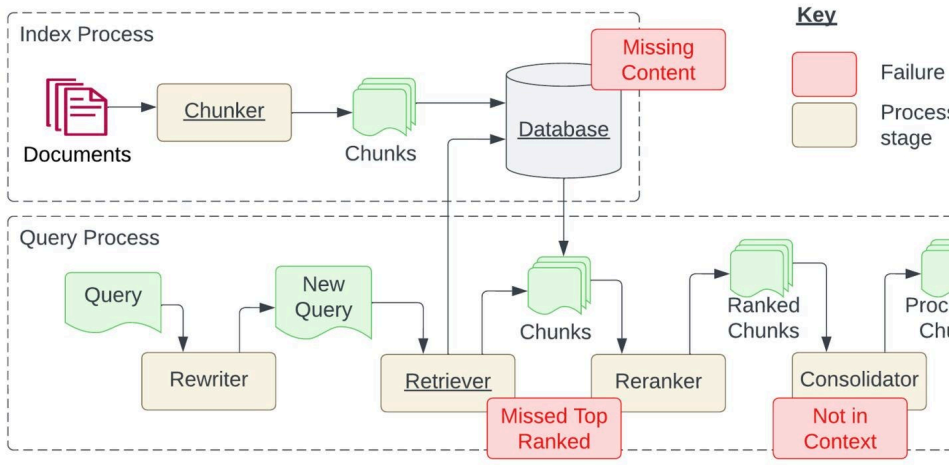


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Gener process is typically done at development time and queries at runtime. Failure points identi boxes. All required stages are underlined. Figure expanded from [19].

Image source: Seven Failure Points When Engineering a Retrieval Augmer

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 of p
as containing conflicting information, no matter h
your RAG pipeline, it cannot do the magic to outp
garbage you feed it. This proposed solution is not
point but all the pain points listed in this article.
prerequisite for any well-functioning RAG pipeline

There are some common strategies to clean your
few:

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

Unstructured.io offers a set of cleaning functiona
library to help address such data cleaning needs.

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 do not sure of the answer," you encourage the model limitations and communicate uncertainty more tr is no guarantee for 100% accuracy, but crafting yo the best efforts you can make after cleaning your

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

TDS is Now Independent!

Pain Point 2: Missed the Top Ranked

Context missing in the initial retrieval pass. The e may not appear in the top results returned by the component. The correct answer is overlooked, ca fail to deliver accurate responses. The paper hinte the question is in the document but did not rank be returned to the user".

Two proposed solutions came to my mind:

Hyperparameter tuning for chunk_size and `simila`

Both `chunk_size` and `similarity_top_k` are paramete the efficiency and effectiveness of the data retrie models. Adjusting these parameters can impact t between computational efficiency and the quality information. We explored the details of hyperpara

both `chunk_size` and `similarity_top_k` in our previous article, Automating Hyperparameter Tuning with LlamaIndex. See the 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` i follows, with `param_dict` containing the parameters `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 runs
fixed_param_dict = {
    "docs": documents,
    "eval_qs": eval_qs,
    "ref_response_strs": ref_response_strs,
}

def objective_function_semantic_similarity(params_dict
    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_strs

    # build index
    index = _build_index(chunk_size, docs)

    # query engine
```

```
    query_engine = index.as_query_engine(similarity_top_k=top_k)

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

    # run evaluator
    eval_batch_runner = _get_eval_batch_runner_semanti
    eval_results = eval_batch_runner.evaluate_response
        eval_qs, responses=pred_response_objs, referer
    )

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

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


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

Reranking

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

- Inaccurate retrieval by directly retrieving the t
  a reranker.
- Accurate retrieval by retrieving the top 10 nod
  `CohereRerank` to rerank and return the top 2 no

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

api_key = os.environ["COHERE_API_KEY"]
cohere_rerank = CohereRerank(api_key=api_key, top_n=2) # return top 2 nodes

query_engine = index.as_query_engine(
    similarity_top_k=10, # we can set a high top_k here to ensure maximum re
    node_postprocessors=[cohere_rerank], # pass the re
)

response = query_engine.query(
    "What did Sam Altman do in this essay?",
)
```

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

TDS is Now Independent!

In addition, you can evaluate and enhance retriev
using various embeddings and rerankers, as detai
RAG: Picking the Best Embedding & Reranker moc

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

Pain Point 3: Not in Context – Consol Strategy Limitations

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

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

Tweak retrieval strategies

LlamaIndex offers an array of retrieval strategies, from basic to advanced, to help us achieve accurate retrieval in Check out the retrievers module guide for a comp retrieval strategies, broken down into different ca

- 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, fine embedding model is a great way to achieve more LlamaIndex has a step-by-step guide on finetunir embedding model, proving that finetuning the em improves metrics consistently across the suite of

See below a sample code snippet on creating a fi 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 ex
answer from the provided context, especially whe
information. Key details are missed, compromisin
responses. The paper hinted: "This occurs when t
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 bac
stress enough the importance of clean data! Do s
your data first before blaming your RAG pipeline.

Prompt Compression

Prompt compression in the long-context setting v
the LongLLMLingua research project/paper. With
LlamaIndex, we can now implement LongLLMLing
postprocessor, which will compress context after
before feeding it into the LLM. LongLLMLingua co
can yield higher performance with much less cos
entire system runs faster.

See the sample code snippet below, where we set up `LongLLMLinguaPostprocessor`, which uses the `longllmlingua` package to run prompt compression.

For more details, check out the full notebook on LongLLMLingua.

```
from llama_index.core.query_engine import RetrieverQue
from llama_index.core.response_synthesizers import Cor
from llama_index.postprocessor.longllmlingua import Lo
from llama_index.core import QueryBundle

node_postprocessor = LongLLMLinguaPostprocessor(
    instruction_str="Given the context, please answer
    target_token=300,
    rank_method="longllmlingua",
    additional_compress_kwargs={
        "condition_compare": True,
        "condition_in_question": "after",
        "context_budget": "+100",
        "reorder_context": "sort",  # enable document
    },
)

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

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

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

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

LongContextReorder

A study observed that the best performance typically arises when crucial data is positioned at the start or conclusion of the input context. `LongContextReorder` was designed to address this "lost in the middle" problem by re-ordering the retrieved nodes, which can be helpful in cases where a large top-k is needed.

See below a sample code snippet on how to defir as your `node_postprocessor` during query engine con details, refer to LlamaIndex's full notebook on `Lor`

```
from llama_index.core.postprocessor import LongContex

reorder = LongContextReorder()

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

reorder_response = reorder_engine.query("Did the autho
```

Pain Point 5: Wrong Format

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

Better prompting

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

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

Output parsing

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

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

LlamaIndex supports integrations with output par offered by other frameworks, such as Guardrails a

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

```
from llama_index.core import VectorStoreIndex, Simple
from llama_index.core.output_parsers import Langchain(
from llama_index.llms.openai import OpenAI
from langchain.output_parsers import StructuredOutputP

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

# define output schema
response_schemas = [
    ResponseSchema(
        name="Education",
        description="Describes the author's educationa
    ),
```

```
    ResponseSchema(
        name="Work",
        description="Describes the author's work experience/background.",
    ),
]

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

# 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 framewo
input string into a structured Pydantic object. Lla
several categories of Pydantic programs:

- **LLM Text Completion Pydantic Programs**: The
process input text and transform it into a stru
defined by the user, utilizing a text completio
with output parsing.
- **LLM Function Calling Pydantic Programs**: The
input text and convert it into a structured obj
the user, by leveraging an LLM function callin

- **Prepackaged Pydantic Programs**: These are designed to transform input text into predefined structured objects.

See below a sample code snippet from the OpenAI [pydantic program] (https://docs.llamaindex.ai/en/stable/module_guides/querying/structured_outputs/pydantic_program.html). For more details, check out LlamaIndex's documentation on the pydantic prog the notebooks/guides of the different pydantic pr

```
from pydantic import BaseModel
from typing import List

from llama_index.program.openai import OpenAIPydanticP

# 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 o
Using the movie {movie_name} as inspiration.
"""
program = OpenAIPydanticProgram.from_defaults(
    output_cls=Album, prompt_template_str=prompt_templ
)

# Run program to get structured output
output = program(
```

```
        movie_name="The Shining", description="Data model for an album."
    )
```

OpenAI JSON mode

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

Pain Point 6: Incorrect Specificity

Output has incorrect level of specificity. The resp necessary detail or specificity, often requiring foll clarification. Answers may be too vague or genera the user's needs effectively.

We turn to advanced retrieval strategies for soluti

Advanced retrieval strategies

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

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

Check out my last article Jump-start Your RAG Pipelines with Advanced Retrieval LlamaPacks and Benchmark with Lighthouz AI for more details on seven advanced retrievals LlamaPacks.

Pain Point 7: Incomplete

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

Query transformations

Comparison questions especially do poorly in naïv
A good way to improve the reasoning capability of
query understanding layer **** – add query trans
actually querying the vector store. Here are four c
transformations:

- **Routing**: Retain the initial query while pinpoin
appropriate subset of tools it pertains to. The
tools as the suitable options.
- **Query-Rewriting**: Maintain the selected tools
the query in multiple ways to apply it across
tools.

- **Sub-Questions**: Break down the query into several smaller questions, each targeting different tools as determined by their metadata.
- **ReAct Agent Tool Selection**: Based on the original query, determine which tool to use and formulate the specific query to run on that tool.

See below a sample code snippet on how to use
Document Embeddings), a query-rewriting techni
language query, a hypothetical document/answer
This hypothetical document is then used for emb
rather than the raw query.

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

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

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

Check out LlamaIndex's Query Transform Cookbo
details.

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

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

TDS is Now Independent!

The above pain points are all from the paper. Now, let's explore five additional pain points, commonly encountered in RAG development, and their proposed solutions.

Pain Point 8: Data Ingestion Scalability

Ingestion pipeline can't scale to larger data volum
ingestion scalability issue in an RAG pipeline refe
that arise when the system struggles to efficientl
process large volumes of data, leading to perform
and potential system failure. Such data ingestion
can cause prolonged ingestion time, system overl
issues, and limited availability.

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

Parallelizing ingestion pipeline

LlamaIndex offers ingestion pipeline parallel proc
that enables up to 15x faster document processin
See the sample code snippet below on how to cr
`IngestionPipeline` and specify the `num_workers` to inv
processing. Check out LlamaIndex's full notebook

```
# load data
documents = SimpleDirectoryReader(input_dir="./data/s

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

```
# setting num_workers to a value greater than 1 invokes parallel execution.
nodes = pipeline.run(documents=documents, num_workers=4)
```

Pain Point 9: Structured Data QA

Inability to QA structured data. Accurately interpreting user queries to retrieve relevant structured data can be difficu complex or ambiguous queries, inflexible text-to- limitations of current LLMs in handling these task

LlamaIndex offers two solutions.

Chain-of-table Pack

`ChainOfTablePack` is a LlamaPack based on the inno table" paper by Wang et al. "Chain-of-table" integ of chain-of-thought with table transformations ar It transforms tables step-by-step using a constra operations and presenting the modified tables to stage. A significant advantage of this approach is address questions involving complex table cells t multiple pieces of information by methodically sli data until the appropriate subsets are identified, effectiveness of tabular QA.

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

Mix-Self-Consistency Pack

LLMs can reason over tabular data in two main w

- Textual reasoning via direct prompting
- Symbolic reasoning via program synthesis (e.g., Python, SQL, etc.)

Based on the paper Rethinking Tabular Data Understanding with Large Language Models by Liu et al., LlamaIndex developed the `MixSelfConsistencyQueryEngine`, which aggregates results from both textual and symbolic reasoning with a self-consis (i.e., majority voting) and achieves SoTA performar code snippet below. Check out LlamaIndex's full r 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 paths
    symbolic_paths=5, # sampling 5 symbolic reasoning
    aggregation_mode="self-consistency", # aggregates
    verbose=True,
)

response = await query_engine.aquery(example["utteranc
```

Pain Point 10: Data Extraction from C

You may need to extract data from complex PDF from the embedded tables, for Q&A. Naïve retriev

data from those embedded tables. You need a better way to retrieve such complex PDF data.

Embedded table retrieval

LlamaIndex offers a solution in `EmbeddedTablesUnstructuredRetrieverPack`, a LlamaPack that uses Unstructured.io to parse out the embedded tables from an HTML document, build a node gra| recursive retrieval to index/retrieve tables based question.

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

TDS is Now Independent!

Notice this pack takes an HTML document as inp| PDF document, you can use pdf2htmlEX to conve without losing text or format. See the sample coc how to download, initialize, and run `EmbeddedTablesUnstructuredRetrieverPack`.

```
# download and install dependencies
EmbeddedTablesUnstructuredRetrieverPack = download_ll
    "EmbeddedTablesUnstructuredRetrieverPack", "./embe
)

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

# run the pack
response = embedded_tables_unstructured_pack.run("Wha
display(Markdown(f"{response}"))
```

Pain Point 11: Fallback Model(s)

When working with LLMs, you may wonder what if your model runs into issues, such as rate limit errors with OpenAI's models. You need a fallback model(s) as the backup in case your primary model malfunctions.

Two proposed solutions:

Neutrino router

A Neutrino router is a collection of LLMs to which queries. It uses a predictor model to intelligently the best-suited LLM for a prompt, maximizing pe optimizing for costs and latency. Neutrino current dozen models. Contact their support if you want to their supported models list.

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

LlamaIndex has integrated Neutrino support throu class in the `llms` module. See the code snippet b 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 Neu
)
```

```
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. It finds the lowest price for any model and offers fallbacks in case th down. According to OpenRouter's documentation, of using OpenRouter include:

> **Benefit from the race to the bottom**. OpenRout price for each model across dozens of provider users pay for their own models via OAuth PKCE
>
> **Standardized API**. No need to change your code between models or providers.
>
> **The best models will be used the most**. Compa often they're used, and soon, for which purpos

LlamaIndex has integrated OpenRouter support th `OpenRouter` class in the `llms` module. See the code 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 insecure outputs, and prevent sensitive information disclosure are all pr every AI architect and engineer needs to answer.

Two proposed solutions:

NeMo Guardrails

NeMo Guardrails is the ultimate open-source LLM offering a broad set of programmable guardrails t LLM inputs and outputs, including content moder guidance, hallucination prevention, and response

The toolset comes with a set of rails:

- **input rails**: can either reject the input, halt fu modify the input (for instance, by concealing information or rewording).
- **output rails**: can either refuse the output, blc sent to the user or modify it.
- **dialog rails**: work with messages in their canc decide whether to execute an action, summo next step or a reply, or opt for a predefined a
- **retrieval rails**: can reject a chunk, preventing to prompt the LLM, or alter the relevant chun

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

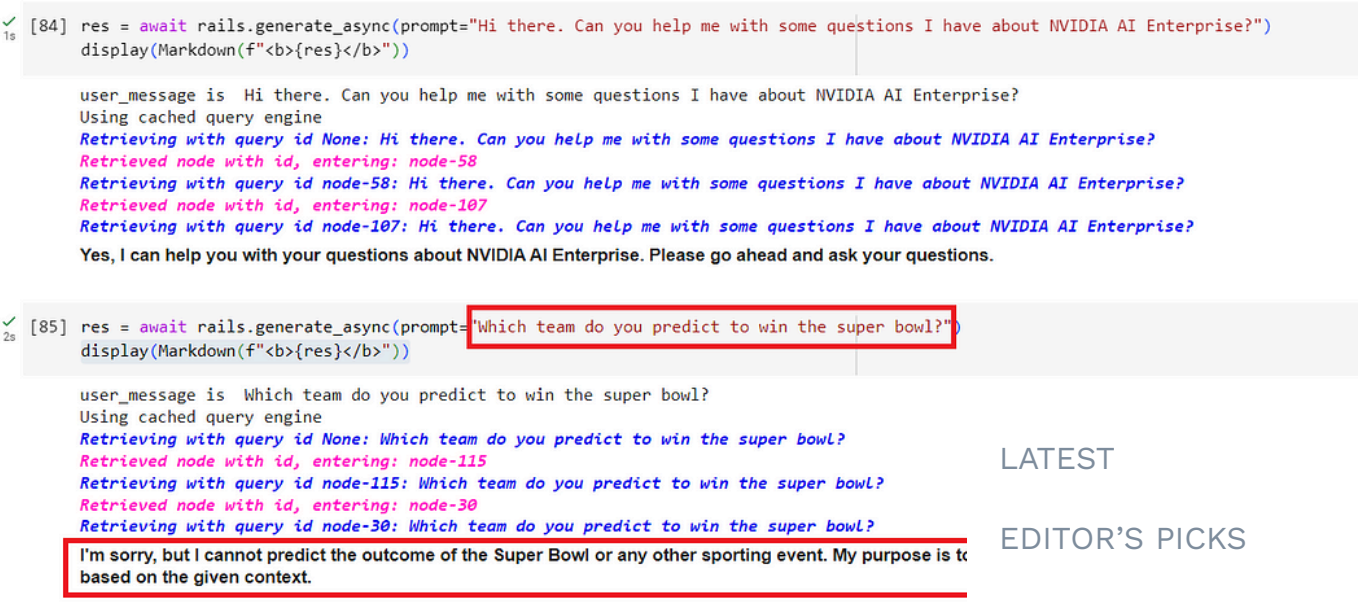
Depending on your use case, you may need to configure one or more rails. Add configuration files such as `config.yml`, `prompts.yml`, the Colang file where the rails flows are defined, etc. to the `config` directory. We then load the guardrails configuration and create an `LLMRails` instance, which provides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

```
from nemoguardrails import LLMRails, RailsConfig

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

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

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



```
[84] res = await rails.generate_async(prompt="Hi there. Can you help me with some questions I have about NVIDIA AI Enterprise?")
     display(Markdown(f"<b>{res}</b>"))

user_message is  Hi there. Can you help me with some questions I have about NVIDIA AI Enterprise?
Using cached query engine
Retrieving with query id None: Hi there. Can you help me with some questions I have about NVIDIA AI Enterprise?
Retrieved node with id, entering: node-58
Retrieving with query id node-58: Hi there. Can you help me with some questions I have about NVIDIA AI Enterprise?
Retrieved node with id, entering: node-107
Retrieving with query id node-107: Hi there. Can you help me with some questions I have about NVIDIA AI Enterprise?
Yes, I can help you with your questions about NVIDIA AI Enterprise. Please go ahead and ask your questions.

[85] res = await rails.generate_async(prompt="Which team do you predict to win the super bowl?")
     display(Markdown(f"<b>{res}</b>"))

user_message is  Which team do you predict to win the super bowl?
Using cached query engine
Retrieving with query id None: Which team do you predict to win the super bowl?
Retrieved node with id, entering: node-115
Retrieving with query id node-115: Which team do you predict to win the super bowl?
Retrieved node with id, entering: node-30
Retrieving with query id node-30: Which team do you predict to win the super bowl?
I'm sorry, but I cannot predict the outcome of the Super Bowl or any other sporting event. My purpose is to
based on the given context.
```


For more details on how to use NeMo Guardrails, article NeMo Guardrails, the Ultimate Open-Sourc Toolkit.

Llama Guard

Based on the 7-B Llama 2, Llama Guard was desig content for LLMs by examining both the inputs (tl classification) and the outputs (via response class Functioning similarly to an LLM, Llama Guard pro outcomes that determine whether a specific pron considered safe or unsafe. Additionally, if it identi unsafe according to certain policies, it will enume subcategories that the content violates.

LlamaIndex offers `LlamaGuardModeratorPack`, enabling Llama Guard to moderate LLM inputs/outputs by 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 interactions with Llama Guard
os.environ["HUGGINGFACE_ACCESS_TOKEN"] = userdata.get("HUGGINGFACE_ACCESS_TC

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

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

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

TDS is Now Independent!

The implementation for the helper function `modera`

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

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

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

        # Check if the moderator's response for output
        if moderator_response_for_output != 'safe':
            response = 'The response is not safe. Plea
    else:
        response = 'This query is not safe. Please ask

    return response
```

The sample output below shows that the query is unsafe and 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, che
article, Safeguarding Your RAG Pipelines: A Step-b
Implementing Llama Guard with LlamaIndex.

Summary

We explored 12 pain points (7 from the paper and
in developing RAG pipelines and provided corresp
solutions to all of them. See the diagram below, a
original diagram from the paper Seven Failure Poi
Engineering a Retrieval Augmented Generation Sy

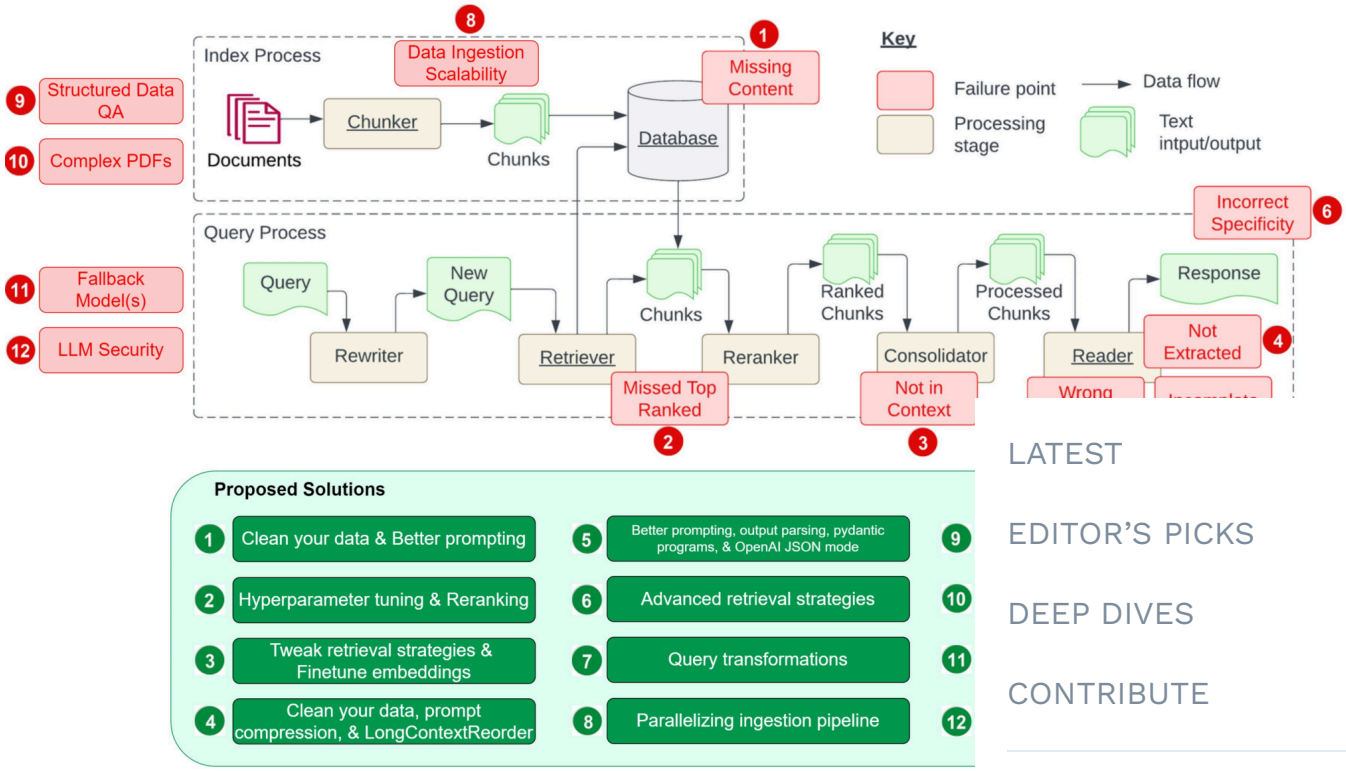


Image adapted from Seven Failure Points When Engineering a Retrieval Augr

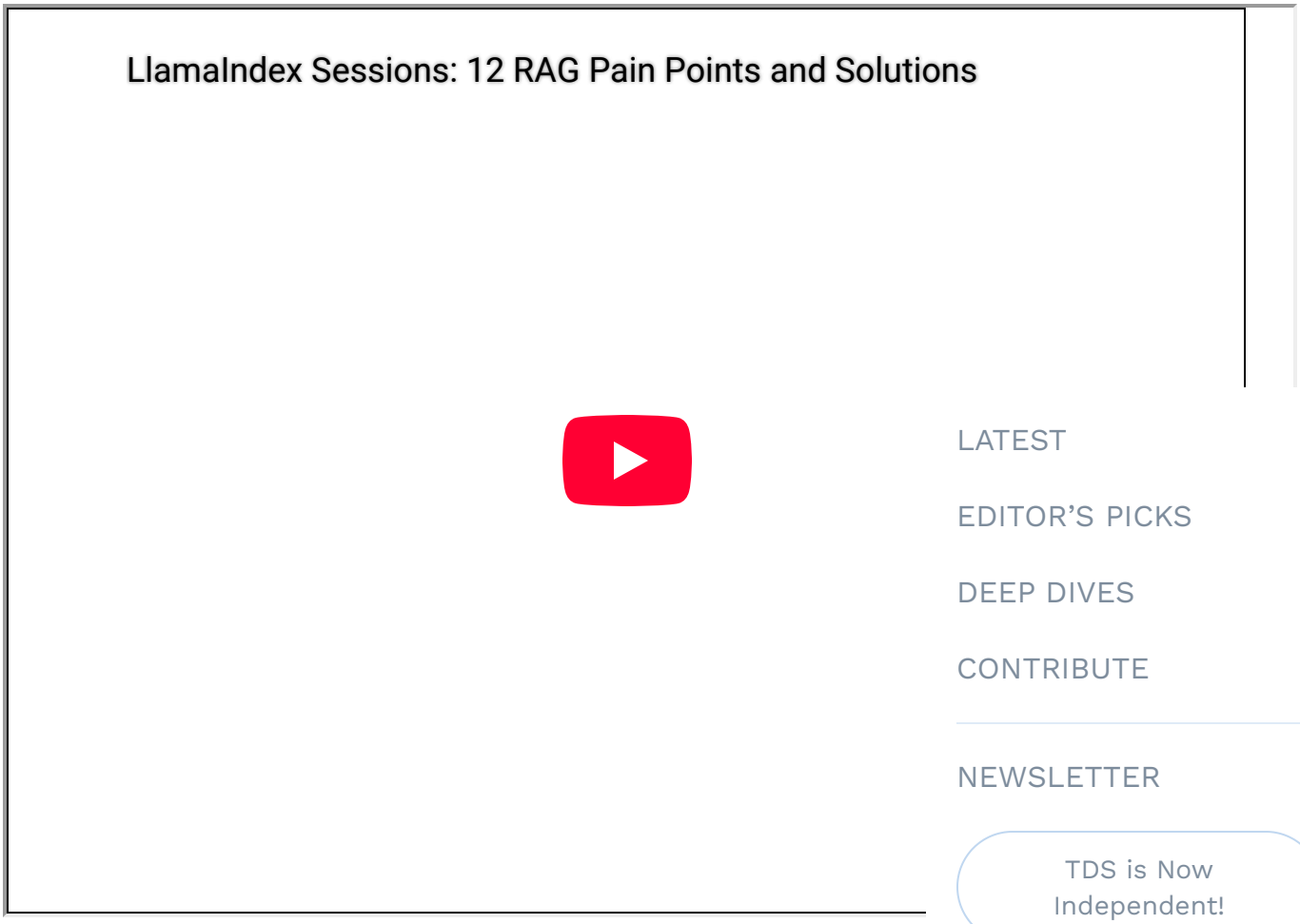
Putting all 12 RAG pain points and their proposed side in a table, we now have:

	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 promptir programs, & Op
6	*Incorrect Specificity (Output has Incorrect Level of Specificity)	Advanced retrie
7	*Incomplete (Output is Incomplete)	Query transforr
8	Data Ingestion Scalability (Ingestion Pipeline Can't Scale to Larger Data Volumes)	Parallelizing ing
9	Structured Data QA (Inability to QA Structured Data)	Chain-of-table p pack
10	Data Extraction from Complex PDFs (Document (PDF) Parsing)	Embedded tabl
11	Fallback Model(s) (Rate Limit Errors)	Neutrino router
12	LLM Security (Prompt Injection etc.)	NeMo Guardrai

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

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

You are also welcome to check out the video vers



Happy coding!

References:

- Seven Failure Points When Engineering a Retr Generation System
- LongLLMLingua: Accelerating and Enhancing I Context Scenarios via Prompt Compression
- LongContextReorder
- Output Parsing Modules
- Pydantic Program
- OpenAI JSON Mode vs. Function Calling for Da
- 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

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

TDS is Now Independent!

WRITTEN BY

Wenqi Glantz

See all from Wenqi Glantz

Topics:

Editors Pick | Llamaindex | Llamaindex Rag | Ll | 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 LEARNIN(

3 AI Use Case Chatbot)

Feature engineer unstructured dat

Shaw Talebi
August 21, 2024 7

LATEST

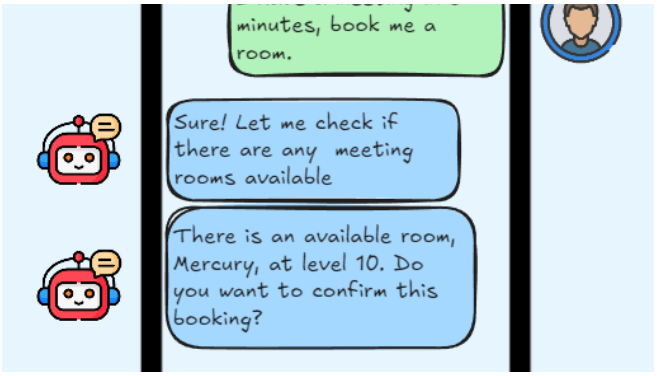
EDITOR’S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

TDS is Now Independent!



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 with Budgete Bandits

With demos, our video

Vadim Arzamasov
August 16, 2024 1

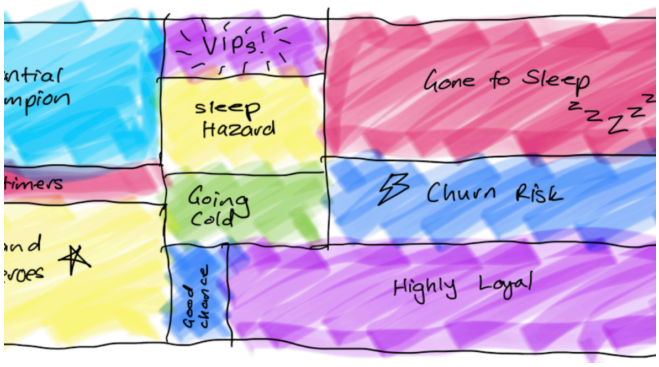
DEEP LEARNING

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



ANALYTICS

Methods for M Lifetime Valu and the Gotch

Part three of a cc guide to CLV tech use-cases

Katherine Munro

November 17, 2023

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

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

DATA SCIENCE

Latest picks: Everyday Ana

Your daily dose o

TDS Editors

November 17, 2020

DATA SCIENCE

Latest picks: No Free Lunch with Feature Bias

Your daily dose of data science

TDS Editors

January 25, 2021 1 min read

Your email*

First name*

Last name*

Job title*

Job level*

Please Select

Company name*

I consent to receive newsletters and other communications from Towards D

Subscribe Now

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

TDS is Now Independent!