Paper ML01

Automated Mock TFL Using a RAG LLM Setup

Nicolaj Thostrup, Novo Nordisk, Aalborg, Denmark

ABSTRACT

Creating Mock TFLs in a study, can be a critical but tedious task. It can be crucial, to enable delegation of efforts but it is also in itself a cumbersome task. In shorter studies it might not yield a great benefit/work ratio. Thus, it is relevant to look into innovative approaches, which can automate some of the work of mock generation, to quickly draft outputs. I will present a proof of concept system, using a LLM and RAG setup, which is designed to improve this process. The system uses a database of historical outputs as input as well as an output title, and then generates or modifies an output according to the requests from the user. The result is an automated approach to mock generation leveraging AI.

INTRODUCTION

The generation of Tables, Figures, and Listings (TFLs) is a fundamental requirement in the conduct of clinical studies, serving as essential components for final study reports. These outputs are typically pre-specified in the form of Mock TFLs to ensure early consensus and alignment across project stakeholders. Depending on the complexity of a given study, the process of drafting Mock TFLs is often iterative and can become labor-intensive, with input from multiple stakeholders introducing diverse perspectives and considerations.

Mock TFLs must adhere closely to the protocol specifications, particularly with respect to the reporting requirements and study objectives. However, their content may also be influenced by elements of study design and project-specific preferences, which can introduce a degree of flexibility. Factors frequently discussed include table orientation, font selection, spacing, decimal precision, statistical representations (e.g., p-values, confidence intervals, means and standard deviations, percentages), MedDRA versions, and footnotes, among others.

Typically, reaching consensus necessitates the creation of multiple iterations of the Mock TFLs. Moreover, these documents are subject to further modification if protocol amendments or related changes are introduced. Taken together, the process is time-consuming and often occurs under tight timelines, as Mock TFLs serve as the foundation for subsequent programming activities and are usually required before programming work commences.

In smaller or shorter-duration studies, the effort invested in generating Mock TFLs may be disproportionate to the overall programming workload. As a result, the creation of detailed Mock TFLs may be deprioritized, potentially leading to reduced documentation or the omission of Mock TFLs altogether.

Consequently, there is a clear need for more streamlined and efficient methods for producing Mock TFLs. Such advancements would facilitate consistently high-quality outputs, improve alignment throughout the project, and enhance the efficiency of downstream programming phases.

CURRENT APPROACHES TO MOCK GENERATION

Current methodologies for Mock TFL generation predominantly involve the manual construction of these documents within platforms such as Word or Excel, wherein practitioners replicate content from previous clinical trials and subsequently adapt appropriate sections and data, including formatting adjustments. This labor-intensive approach is susceptible to errors inherent in manual processes and presumes comprehensive familiarity with the breadth of historical outputs generated by the organization. Consequently, efficiently leveraging prior work and identifying suitable templates for reuse can present significant challenges, particularly for individuals with limited experience, thereby resulting in considerable time expenditure and suboptimal efficacy.

Some organizations maintain standardized libraries of generic mock templates, which require only minor customization and the insertion of study-specific information into predefined placeholders. Alternatively, proprietary in-house tools may be employed to streamline layout concerns, enabling users to input and update relevant data with the possibility of minor manual post-processing to refine formatting and structural elements, depending upon the tool's capabilities. While these

solutions provide partial relief from manual effort and expedite certain aspects of the process, the repetitive nature persists in scenarios where substantial human intervention remains necessary.

In instances where no pre-existing Mock TFL is suitable for reuse, it may be necessary to consult external repositories such as clinicaltrials.gov to locate Statistical Analysis Plan (SAP) reports or results from analogous studies, with the hope of identifying a compatible template.

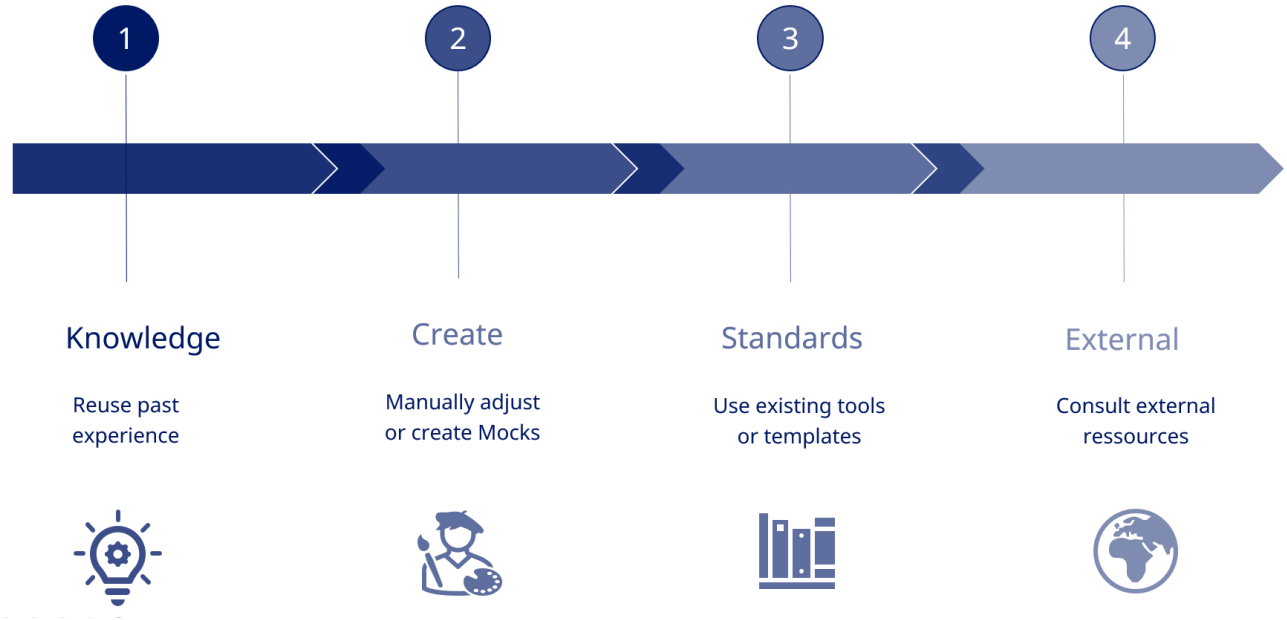


Figure 1 - Mock TFL creation steps

The automation of mock TFL generation holds considerable promise for enhancing both efficiency and precision. Advanced technologies, such as large language models (LLMs), offer the potential to expedite the production of high-quality mock TFLs while substantially reducing the required human effort and turnaround time.
Manual generation of mock TFLs potentially demand multiple weeks of dedicated labor, with further extensions possible due to iterative stakeholder feedback and subsequent modifications, not accounting for additional changes or the development of new Mock TFLs at later stages. Reducing the cycle time for incorporating stakeholder input and updating Mock TFLs can substantially benefit project momentum, facilitating prompt decision-making and execution. Accelerating the initial drafting phase through automation therefore represents a significant opportunity for process optimization.

PROOF OF CONCEPT LLM AND RAG SOLUTION

Leveraging the rapid advancements in generative artificial intelligence presents a practical avenue for addressing some of the immediate inefficiencies associated with mock TFL production. Many of these challenges do not necessitate groundbreaking technological innovation, but rather, can be addressed through the systematic organization and application of existing data and established methodologies.

By employing Shiny for Python as the user interface, integrating large language models for automated content generation, and utilizing historical outputs as foundational reference material, it is feasible to construct a proof of concept (PoC) that demonstrates meaningful efficiency gains through automation.

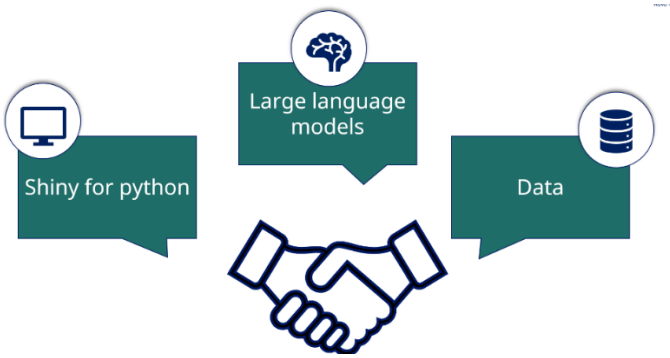


Figure 2 - Components of the application

In the PoC application, Shiny for Python serves as the interactive user interface, large language models drive automated content generation, and historical data provides essential reference material for accurate mock TFL outputs.

The overall architecture of the application is illustrated in Figure 3 below.

- **User Input:** A search query or mock edit query is submitted via the Shiny frontend interface.
- **Embedding Query:** The input is converted into an embedding and sent to the **vector database**.
- **Vector Search:** The vector database retrieves **relevant outputs** (contextual information).
- **Context Injection:** Retrieved outputs are passed as context to the **LLM** for enhanced generation.
- **Mock Generation:** The LLM processes the context and generates or updates mock outputs.
- **Response:** The outputs are displayed as responses on the frontend for user validation or further edits.

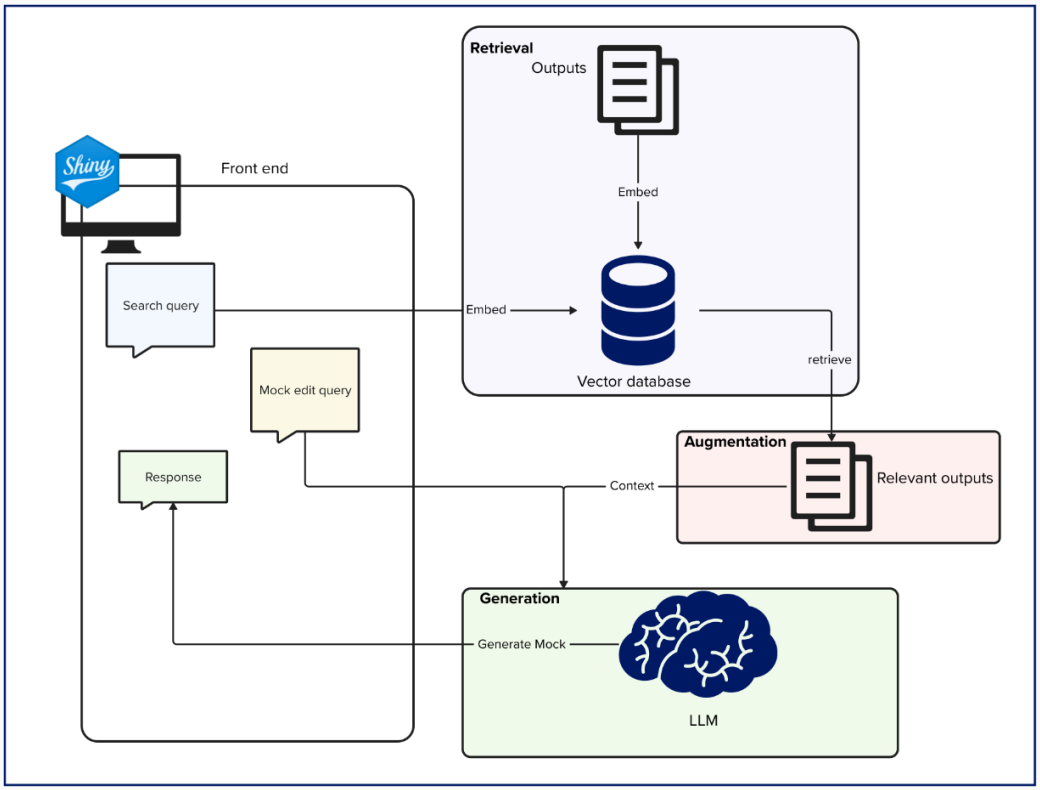


Figure 3 - Architecture of application

RETRIEVAL - DATABASE OF HISTORICAL OUTPUTS

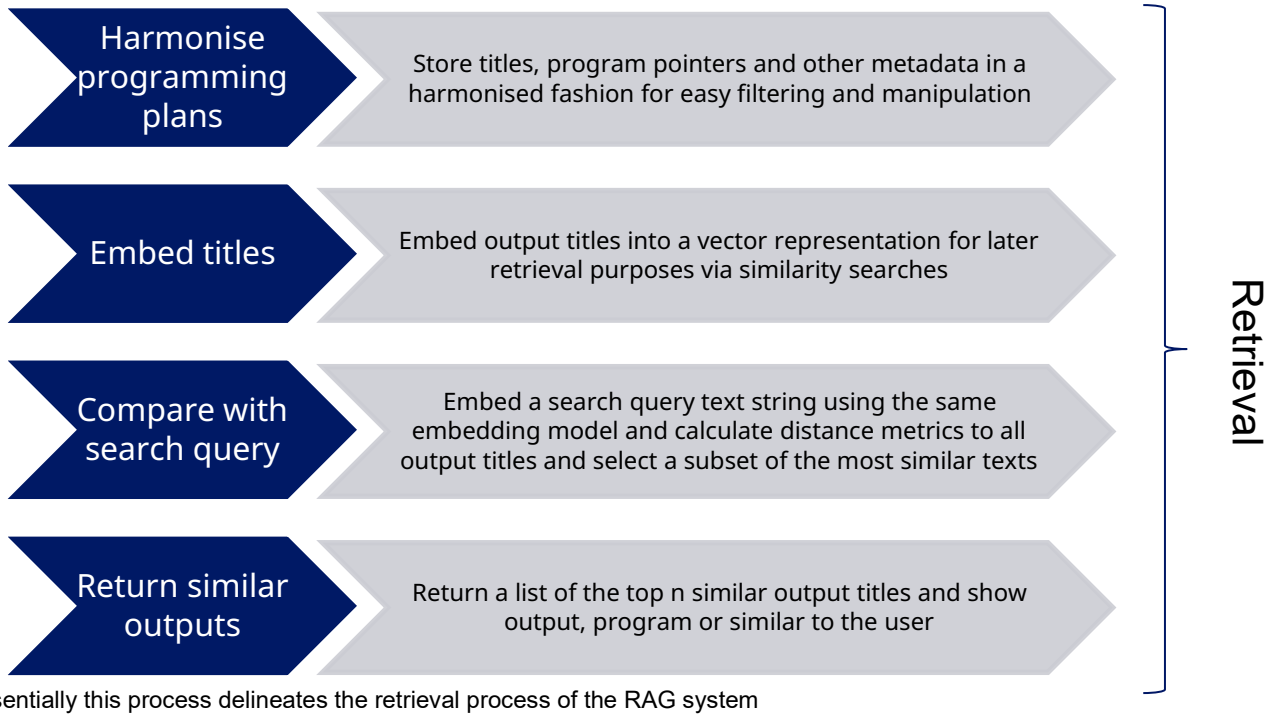
All previously generated outputs and Mock TFLs are systematically archived within Novo Nordisk. The harmonization of multiple source repositories, or centralized access to a singular resource, facilitates efficient indexing and expedites retrieval processes.

Within the proof of concept, an extensive dataset comprising programming plans, each linked directly to specific outputs, is utilized. This repository encompasses 72,547 titled outputs and 6,253 unique programs derived from nearly 300 distinct clinical trials. Such a database serves as the foundational component for the retrieval and reutilization framework described herein.

Each output title is transformed into a vector representation via an embedding language model (OpenAI text embedding ADA), enabling advanced retrieval mechanisms. The conversion of textual data into vector space representations is a standard technique in Retrieval Augmented Generation (RAG) systems, as it allows for the computation of distance metrics and the derivation of similarity scores—most notably cosine similarity. This approach permits the identification of top matches for a given query by comparing the embedded search vector to existing title vectors. Given the continual improvements in embedding models, this methodology proves highly effective for the retrieval of semantically related textual segments.

Applied to TFL outputs, this technique demonstrates considerable efficacy in locating relevant outputs in response to user queries. The flexibility inherent to embedding-based retrieval systems accommodates variations in phrasing and even corrects for minor typographical errors. For instance, a query such as “Baseline demograpics” successfully returns outputs related to Baseline demographics summaries, while “disposition summary” yields “subject disposition summary” outputs.

The retrieval process is schematically illustrated here:



Essentially this process delineates the retrieval process of the RAG system

AUGMENTATION AND GENERATION

After retrieving a set of relevant outputs with corresponding metadata, the next step in the R**A**G system is to augment the retrieved context with the user query.

In this proof of concept, the augmentation is a simple concatenation of the top matches to form a varied and relevant context for the mock generation.

The RA**G** generation step is then tasking the LLM with generating a representative Mock TFL based on the context. Hereafter it is possible for the user to query the LLM to modify or change the Mock TFL, according to specific requests.

Examples could be:

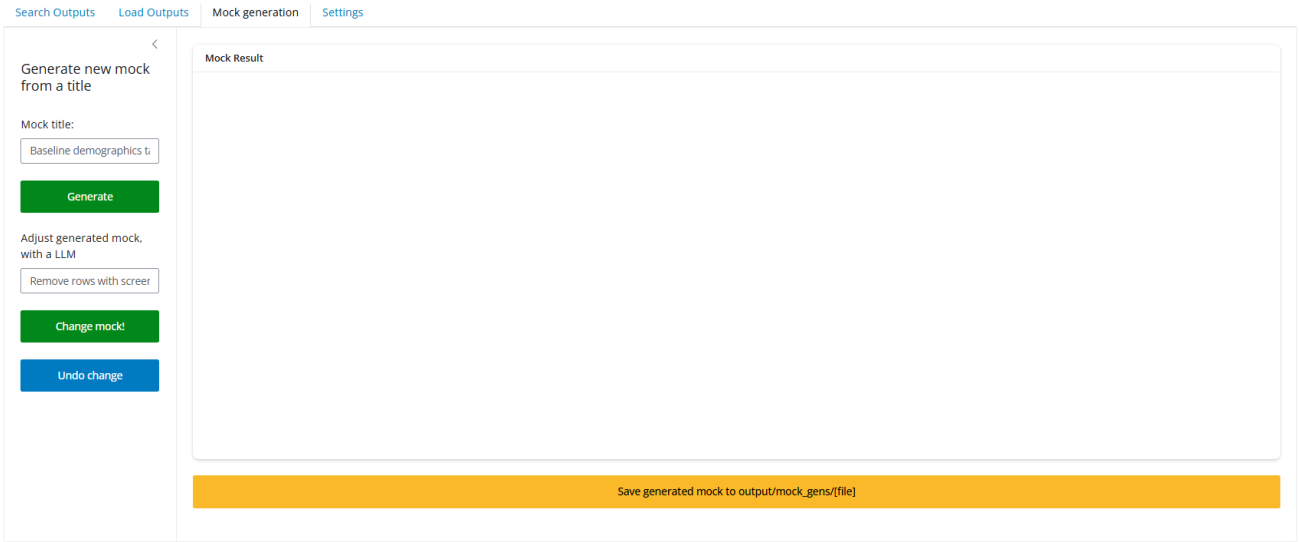
- “Add an additional treatment arm to the output called Drug A”
- “Add an additional data collection point for Bilirubin at visit 10”
- “Rename all visits according to this schema: Visit 1 day 1 Visit 1 day 2 Visit 2, Visit 3 ...”
- Add an endpoint entry for [ENDPOINT] and update footnote accordingly. In the header replace "FAS" with Full analysis set and "CI" with Confidence interval. BE VERY CAREFUL on spacing and alignment for the columns. Rename title to “PK endpoints for [DRUG]”

In the context of this case study, the augmentation and generation step serves as the dynamic engine moving the RAG workflow forward. Here, the system takes the enriched context and integrates it with the original user query. This composite prompt is then provided to the generative model, which synthesizes a new, bespoke output: a modified TFL mock-up that accurately reflects both the retrieved data and the user’s initial query. Through this iterative dialogue between retrieval, augmentation, and generation, with optional additional modifications, the process ensures both flexibility and efficacy, allowing automation in the Mock TFL creation phase aligned with evolving user needs.

SHINY FOR PYTHON

To realize the practical benefits of the RAG system, integrating it with a user interface is essential—and Shiny for Python offers a great framework for this purpose. By building an interactive Shiny app, the dynamic capabilities of the RAG workflow become accessible to users, who can modify, review, and iterate on TFL mock-ups directly in their browser. Such an application streamlines the process, making advanced automation approachable without demanding technical expertise from the user. However, importantly the solution must be coupled with a publication and access platform, such as Posit Connect. With Posit Connect, it is possible to securely deploy, manage, and share applications, ensuring that the RAG-powered solution is accessible to the end-user.

The UIs main tab is the “Mock generation” tab which conducts the RAG of a Mock TFL. However for usability purposes it is also possible for the user to retrieve specific outputs in the “Search Outputs” tab, to get inspiration and review different output layouts. The “Load Outputs” tab is for retrieving a specific output with the intention of modifying it.
Thus, in essence the RAG approach has been “split up” to allow for a more flexible approach.



RAG

Retrieval Augmented Generation (RAG) is an innovative paradigm in the field of artificial intelligence and natural language processing. It marks an evolution from traditional large language models (LLMs) by introducing a dynamic retrieval mechanism that supplements the model’s responses with relevant, up-to-date information from external sources. This blend of retrieval and generation empowers systems to produce more accurate and contextually relevant outputs. [3]

Conventional LLMs, rely heavily on the static information encoded within their training data. While these models can exhibit fluency and general knowledge, they are in deep water when confronted with specialized queries or company specific requests. RAG addresses this limitation by enabling models to "look up" supporting information on demand, much like a human referencing a library or database.

While the RAG framework offers potential, its implementation is not without drawbacks:

- Retrieval Quality: The accuracy and usefulness of generated responses hinge on the effectiveness of the retrieval system. Poorly retrieved data can mislead the generative model.
- Latency: The additional retrieval step can introduce delays, especially when querying large or remote data sources.
- Context Window Limitations: Language models have finite input length, necessitating careful selection and summarization of retrieved outputs. Albeit models nowadays really are improved on the context window length, and it is to the discretion of the programmer to curate the retrieved knowledge appropriately rather than "dumping everything" uncritically.
- Evaluation Complexity: Assessing the correctness and relevance of RAG outputs can be more intricate, given the interplay between retrieval and generation. But much can be combated via careful prompt engineering.

LLM

Large Language Models (LLMs) have undergone significant transformation since their inception. Early iterations were limited by smaller datasets, modest computational resources, and simpler architectures that constrained their capacity for nuanced understanding. Over time, advances such as much larger pre-training corpora, and more refined fine-tuning methods have propelled LLMs to new heights in both comprehension and generation of natural language.
Improvements include:

- Scale: Modern LLMs utilize billions (or trillions) of parameters, allowing for richer representations and more sophisticated inferences.
- Contextual windows: Extended context windows let these models consider longer and more complex documents, resulting in more coherent and contextually accurate responses.
- Multimodality: The newest generations can process not just text, but also images, audio, and other data types, broadening their applicability.
- Efficiency and Latency: Architectural optimizations and hardware acceleration have substantially reduced response times while increasing throughput.

The result is a class of AI models that can answer increasingly complex questions, generate content more precisely, and engage in meaningful dialogue across diverse domains. The advancement is illustrated in below table, where GPT 3.5 turbo, a model released in Nov 2022, can be compared with newer models from 2024-2025.

Model	**Context**	**GPQA Score (%)**	**MMLU Score (%)**	**SWE Bench (%)** [2]	**Multimodal**	**Release**
GPT-5	400.000	85.7	92.5	74.9	Yes	Aug 2025
GPT-4.1	1.047.576	66.3	90.2	55.0	Yes	Apr 2025
GPT-4o	128.000	70.1	85.7	31.0	Yes	May 2024
GPT-3.5 turbo	16.385	30.8	69.8	-	No	Nov 2022
Claude Sonnet 4	200.000	75.4	-	72.7	Yes	May 2025
Deepseek R1	131.072	71.5	90.8	49.2	No	Jan 2025

[1] [2]

These improvements are also highly relevant in the context of this solution. Older models have a hard time comprehending the important layout elements of the context and are more prone to producing outputs which are too far from the target. On the other hand, the more recent and advanced models really seem to have a great ability to stick to the nature of the context and implement adjustments seamlessly into the original design.

CONCLUSION

The advent of advanced language models presents significant opportunities for optimizing the utilization of historical data and accumulated domain knowledge. This proof of concept demonstrates that previously inaccessible insights can now be readily surfaced, enabling a comprehensive examination of archival information for the purposes of innovation and informed decision-making.
Notably, state-of-the-art LLMs facilitate the rapid adaptation of existing TFLs, allowing them to serve as effective Mock TFLs for subsequent trials. This increased agility in design iteration reduces time constraints and fosters greater experimental precision.

However, the present approach remains constrained in terms of full automation. While the generation of individual Mock TFLs has been streamlined, the capacity to automate the production of entire sets of Mocks from a singular set of instructions has not yet been fully realized. Achieving this objective demands a robust mechanism for providing LLMs with a holistic understanding of the target trial, as well as accommodating project-specific requirements. The complexity is compounded by the fragmented nature of relevant information, which is typically distributed across multiple platforms and sources.

Looking forward, it is reasonable to predict that the boundary between manual creation and automation will continue to blur. As LLMs become more contextually aware and data ecosystems grow increasingly interconnected, researchers will be empowered to explore broader, deeper, and more creative avenues—potentially redefining standards for knowledge management. The journey toward fully autonomous, context-sensitive automation is ongoing, and the future likely entails automated approaches to generate programs producing the actual outputs rather than just the Mock TFLs.

REFERENCES

[1] https://llm-stats.com/
[2] https://www.vellum.ai/llm-leaderboard
[3] Gupta, S., Ranjan, R., & Singh, S. N. (2024). A comprehensive survey of retrieval-augmented generation (RAG): Evolution, current landscape and future directions. arXiv. [https://arxiv.org/abs/2410.12837]

RECOMMENDED READING

Get started with your own RAG application using HuggingFace articles
https://huggingface.co/blog/ngxson/make-your-own-rag

Understanding Retrieval Augmented Generation, Medium:
https://felixvidalgu.medium.com/understanding-retrieval-augmented-generation-659093931289

The original paper on RAG by Meta AI researchers:
Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems, 33, 9459-9474.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Nicolaj Thostrup
Company: Novo Nordisk A/S
Email: ncjt@novonordisk.com