Evaluation of Large Language Models for an AI Chat Assistant Focused on Pumas and Pharmacometrics

Agastya Deepak Vinchhi Juan José González Oneto
Michael Hatherly Vijay Ivaturi

Abstract

The Large Language Model (LLM) landscape is constantly changing, with new models emerging rapidly and outperforming established benchmarks. For professionals working on LLM applications, this means constantly being aware of top-performing LLMs that can replace current LLMs in their internal AI applications; an LLM that is considered the optimal choice in a scientist's current AI architecture cannot be assumed to remain the optimal choice as models constantly evolve. At PumasAI, while building AskPumas, a Retrieval Augmented Generation (RAG) based AI chat assistant focused on Pumas and pharmacometric-related inquiries, we noticed the importance of finding a structure to assess numerous LLMs. The motivation behind this research is to establish a framework to evaluate LLMs and select and rank them based on different criteria. This paper will explore the methodologies of our evaluation criteria, how this has served in helping select the ideal LLM for AskPumas, and how our established framework can serve as a guide for developers who seek to find the optimal LLM for their domain-specific AI application. Using this framework, we identified `gpt-5-chat`, `gemini` models, `qwen3-max`, and `grok-4-fast` as top-performing LLMs for AskPumas. The differences among the highest-ranked models are small (~1-2%), suggesting that we identified sets of strong candidates for AskPumas, rather than a single definitive winner. Overall, we conclude that model selection for domain RAG applications should be treated as a modular process that considers trade-offs between metric weights and insights from embedding-based clustering, so AskPumas can adapt its priorities as the LLM landscape evolves.

1 Introduction

At PumasAI, the AI assistant AskPumas is a highly extensible RAG (Lewis et al. 2021) system, focused on helping users of Pumas™ (Rackauckas et al. 2020) and DeepPumas™ (Korsbo et al. 2023) perform their work more efficiently. A RAG system enhances performance tasks by utilizing curated private sources, unlike LLM training data, which is static and cannot access proprietary data. This enables AskPumas to have accurate outputs representative of topics from the provided data. For this context, we have developed a structured methodology tailored specifically for a RAG system that helps us confidently determine ideal and reliable LLM choices suitable for AskPumas. Since each AI application has its own requirements and needs, scientists can adopt our reproducible methodology to evaluate and select LLMs with their own use case, architecture, and constraints.

Choosing the correct LLM for a RAG system should not be an arbitrary selection. There are a range of factors to consider, such as size, purpose, license, and implementation. For instance, some LLMs can be self-hosted, which ensures compliance and data privacy, whereas API based LLMs are easier to integrate and can be a more cost-efficient option (see Section A). Having established these trade-offs, our standardized benchmarking allows us to directly compare LLMs and rank them against each other.

However, existing evaluations and benchmarks such as Holistic Evaluation of Language Models (HELM) (Liang et al. 2023) and Massive Multitask Language Understanding Pro (MMLU-Pro) (Wang et al. 2024) do not explore the abilities of LLMs in the context of the RAG system. LLMs in RAG need to effectively identify key information from retrieved contexts. For instance, noise robustness, negative rejection, information integration, and counterfactual robustness are fundamental abilities required in RAG, emphasized in previous work (Chen et al. 2023), which are not considered in generalized benchmarks. Beyond these general benchmarks, recent work such as Comprehensive RAG Benchmark (CRAG) focuses on a comprehensive RAG benchmarking, further highlighting the need for RAG-specific evaluation benchmarks (Yang et al. 2024) and systemic reviews of RAG systems to enhance domain specific knowledge in LLMs (Murtiyoso, Tahyudin, and Berlilana 2025). Our contribution poses a methodology tailored to a RAG system for private pharmacometric data to identify top-performing LLMs for AskPumas.

2 Methodology

In this section, we explore the methodology used to evaluate various LLMs for a RAG-oriented system. We began by defining fixed factors to select various LLMs and proceeded by building a domain-specific prompt repository, which was used to evaluate LLMs on fixed criteria. By considering both quantitative and qualitative metrics, our pipeline tackles LLM selection holistically, highlighting what scientists should focus on for selecting an LLM.

2.1 Exploring LLMs

When exploring different LLMs for a RAG application, it is important to understand the needs of the architecture. There are numerous LLMs to choose from; when making an extensive list, having fixed factors can help select suitable LLMs based on the scope and needs of the architecture. For example, for the AI assistant AskPumas, we prioritized chat models that can be accessed via an API (such as `gpt-5-chat` and `claude-sonnet-4.5`) and some models that have the option to be self-hosted as well (such as `mistral-small-3.2-24b-instruct` and `gemma-3-12b-it`); as a scientist, it is crucial to select the right model, based on your use case, computing resources, and budget.

To analyze prospective LLMs, we created a wide range of fixed factors that need to be considered, based on the LLM's abilities, feasibility, practicality, and robustness. Having fixed factors allows a scientist to have a strong reason to select a specific model over a different one. Using these factors as a guide, we selected 19 different LLMs that match our system's needs - having a comprehensive method allowed us to select the correct model with confidence and methodical rigor:

Model Size. The model's size in terms of parameters can significantly affect its overall performance. Larger models can have the capacity to interpret complex query inputs, though, it can be computationally costly. On the other hand, smaller models can sacrifice performance for better efficiency. It is important to compare the trade-off between the size of the model and its performance - this heavily depends on the scope of the scientist's application.

Implementation. Examining the ease of integrating the model for both benchmarking and implementation can help understand the potential of the model and how it can fit into the scientist's RAG architecture. For instance, LLMs can be implemented through applications and libraries such as LangChain (LangChain, n.d.) or Hugging Face (Hugging Face, n.d.); such models are easy to implement into already existing Python pipelines. During the output generation phase, we had to run 17 LLMs on 181 prompts. Every LLM had its own implementation method, whether it was unique API keys or running models locally. We needed to carefully consider each implementation method to develop a script to run all LLMs at once smoothly. If a scientist has privacy concerns with their RAG application, then they should avoid API based models; self-hosted solutions will work better in this situation. However, integrating an API-based model like OpenAI's `gpt-5-chat` can be much easier than deploying a model locally or hosting a model through Hugging Face or running it through external applications like Ollama (Inc., n.d.). These are examples of different ways to consider implementation as a factor when choosing the right LLM for your RAG model.

Licensing. This is to review licensing terms associated with the model, including any significant restrictions on model use and modification. Further, looking into a specific LLM license – to see if the model can be used commercially – is another factor to consider. Some LLMs restrict commercial use, whereas others will allow you to integrate LLMs into your application. This factor is especially critical for self-hosted models, where licenses typically place explicit limits

on commercial use, in contrast to API based LLMs that typically allow commercial use but apply usage-based pricing for API calls. As a scientist, if you are creating an application as a product, for instance, a business-to-business product, then you will need to find LLMs that have licenses that allow you to do the same. It is important to check what can be done and what cannot be done to avoid any repercussions.

Purpose. Different models are created for different purposes, for instance, Code Assist, Generative AI, Conversational AI, Language Support, etc. The purpose of different models can significantly alter the use case of the language model. If you aim to build a standard RAG model or plan to use an LLM to develop some form of a chatbot or a task-specific automation tool, you will lean towards using a Generative AI LLM. There is a range of other use cases. For example, a scientist who needs to classify different images for data analysis would use an image feature extraction LLM, or a scientist who would like to gain insight into their code base would rather choose a code assist LLM. When we were selecting LLMs, we wanted to create a chatbot for pharmacometrics queries. So, we chose Generative AI LLMs in our list of potential LLMs. This factor allows you to choose the correct LLM and adapt according to your application's needs.

Beyond the fixed factors above, there are additional variable factors to consider, whose importance depends on your AI application and use case. These are more context-dependant:

Pre-trained data. This factor focuses on the training set of the LLM. This can range from internet text, different research papers, curated datasets or other relevant sources. The quality of this data heavily influences the language model's accuracy. It is integral to understand where the data comes from and how it can play a role in the model's output. LLMs are trained in a range of different areas for diversity; it is good practice to see where the training data is coming from and how it can impact the outputs of the LLM. LLMs can also be fine-tuned on specific curated datasets or other specialized domains, which can result in high-quality outputs, depending on the context of the data. When selecting your final LLM list to potentially use and integrate it is informative to know the kind of data the model is trained on.

Cost. Depending on what your budget is for your application, the cost of deployment is a factor to consider as well. Constantly making API calls can be very expensive, especially when building out an application that requires heavy use of LLMs. Self-hosting an LLM can be more expensive, as hosting such LLMs is computationally expensive and requires resources, whether hosting on a server or local GPU infrastructure. Self-hosting LLMs also requires maintenance and scaling costs such as GPU upgrades, energy costs, and opportunity costs (since scientists will have to manage the infrastructure, rather than other tasks). You must find the optimal cost tradeoff between your AI application's budget and the model's performance and capabilities.

2.2 Data, Privacy, and Compliance

A cause of concern for most scientists who integrate LLMs into their workflow or software product is the privacy of their data. Making API calls to LLMs that gain access to your data

can be a major compliance concern for the company. For instance, if you have important customer data that cannot be shared outside of the company due to compliance rules, it is unwise to use an API based model that has access to the data you provide. We were concerned with exactly that and decided to evaluate self-hosted LLMs as well. For instance, `mistral-small-3.2-24b-instruct` and `gemma-3-12b-it` were top contenders for AskPumas; privacy and compliance concerns are not an issue anymore when self-hosting your LLM on your own servers (or GPUs). This ensures that your application is well within the limits of compliance rules and does not violate any privacy or data leaks (Shanmugarasa et al. 2025).

There are certain situations where it can be practical to use API based models if the context fits the specific trade-offs of the application. For example, consider a situation where a scientist intends to incorporate an LLM to create an application for workflow automation within the company; self-hosting an LLM has a cost associated with it – it can be expensive to host and requires effort to integrate and maintain into the system. In such a situation, where privacy is not a concern, it is preferable to use API LLMs since these services are optimized for these use cases. Nevertheless, it heavily depends on the use case of what you are building as a scientist, although it is crucial to consider the sensitivity of your data.

2.3 Establishment of a Prompt Repository

Before implementing the benchmarking pipeline, we assembled a curated collection of 181 prompts representing common pharmacometric-related questions. Having an established prompt repository helps us compare the outputs of numerous LLMs in a more standardized approach. Since each LLM gives a different output in a unique way, creating a prompt repository of 181 prompts enabled us to systematically evaluate and benchmark model responses across different scenarios.

We aimed to evaluate our chosen LLMs based on practical questions related to pharmacometrics, PumasAI products, or Julia that users at PumasAI might have. To achieve this, we recorded and transcribed interviews with six expert scientists at PumasAI, who helped us identify different kinds of prompts. We instructed them to generate both qualitative and quantitative prompts to create diversity in our prompt repository. Asking different scientists with different specialties and thought processes helped make our prompt repository deep and comprehensive.

We found that to surf through all our 6 recorded transcribed interviews to pinpoint specific prompts was extremely time-consuming. Further, to streamline this process, we employed an LLM to help us extract prompts directly from the transcribed interviews. By running Python scripts in combination with OpenAI's `gpt-4o`, we extracted over 600 potential prompts. To account for variability, we ran the model multiple times with different temperature settings: 0, 0.05, 1.0, 1.15, and 2.0. Changing this hyperparameter generated unique questions based on the insights from the interviews, making our prompts diverse.

From these 600 prompts, we narrowed our prompt repository to 145 unique prompts through the process of elimination. This involved carefully selecting prompts that create a well-balanced

prompt repository. Our choice of exactly 145 prompts was not intentional; rather, it was a result of our systematic manual filtering process for relevance in the pharmacometric domain. Within this set of 145 prompts, we also included a subset of 14 questions, where the retrieved context does not have the information required to answer the question. These 14 prompts are unaswerable by design to test answer rejection of the LLM, which is the model's ability to refuse to answer when complete or correct information is not present in the retrieved context provided (see Section 2.8).

In addition to the prompts generated from interviews, we also leveraged real prompts posed by PumasAI users to AskPumas. We first anonymized all logged prompts by removing any notable user identifiers and sensitive information, ensuring that no user data was visible. From this anonymized pool of prompts, we carefully selected 36 unique prompts that were relevant to incorporate into our prompt repository. Utilizing these real-world questions in our repository made it comprehensive in nature, as it was representative of how AskPumas is currently being used by users.

Combining the 145 interview based prompts with the 36 PumasAI customer prompts yielded a total of 181 prompts which is our final repository. Here, it is essential to select prompts that are an accurate representation of user queries that will be posed to your AI application or the types of LLM invocations the application will make; this is because the generated outputs of the prompt repository will be subsequently evaluated. For a scientist, creating a domain-specific prompt repository is crucial as it will contextualize the subsequent evaluation, highlighting how your LLM will perform in your AI application.

2.4 Human Vetting

After the selection of 19 models in the exploration stage, we decided to manually verify whether the output of all these LLMs is up to our standards for AskPumas. By running these models individually, we gauged the overall strength of the LLM and its potential to answer domain-specific questions in pharmacometrics. The primary purpose of the human vetting process allowed us to filter out prospective models that had little to no potential early on, before proceeding with more rigorous evaluation methods. Doing this initial step helps eliminate models that output low-quality results or have significant restrictions or limitations.

During this process, we ended up eliminating a total of 2 LLMs, resulting in a shortened shortlist of 17 models (see Section A). Since AskPumas is a RAG-based architecture, some of the 19 LLMs in our shortlist were unable to perform adequately in this setup. When considering models for your own AI application before evaluation, consider using this vetting process as a tool to help understand what kind of LLM you are looking for. By understanding the requirements you need from an LLMs you will be more aware of the limitations of different LLMs, helping you make a more informed decision in your shortlist. Finally, you can eliminate LLMs that do not adhere to the quality and needs of your AI application. The two models we eliminated are Meta's `llama-3.3-8b-instruct` and Microsoft's `phi-4` (see Section A). The

former required some outputs to have authentication and approval from Meta's end, which complicated our evaluation. The latter quickly capped off input token limits from the relatively large retrieved context we provided, showing that it cannot handle our large context window.

2.5 Generating LLM Outputs

Before formal evaluation, we generated outputs of our shortlisted 17 LLMs for all 181 prompts. Since our LLM choices were diverse, this posed a challenge – running LLM outputs individually for all 181 prompts - each LLM had its own method of invocation. This complication encouraged us to create a unified Python script that streamlined the generation of our outputs across all LLMs. To execute this, we ran our Python scripts incorporating a service that enabled to call multiple LLM APIs parallelly, enabling us to run multiple large API based models in our LLM shortlist. This method successfully generated ~3000 outputs of all our models, which were organized and stored methodically in a CSV file.

2.6 RAGAS (Retrieval Augmented Generation Automated Scoring) – Automated Model Evaluation

After generating all LLM outputs, we proceed with formal evaluation. Our first evaluation criterion employs RAGAS (Es et al. 2025), a Python library that provides strong tools helping users evaluate their RAG models. This automated method generates a quantitative score on a scale of 0 to 1, evaluating the LLM's fairness and accuracy. By utilizing RAGAS (Es et al. 2025), we obtained an objective method of comparison across all 17 LLMs, providing our evaluation pipeline with consistent and standardized measurements for comparison.

RAGAS offers numerous tools for evaluation; we chose two metrics that were the most relevant to us: **Faithfulness** and **Answer Relevancy**.

2.6.1 Faithfulness

Faithfulness measures how closely the claims made in the LLM's outputs align with the retrieved context from the RAG application. Higher faithfulness scores demonstrate that the LLM output accurately depicts the retrieved context. Lower scores demonstrate the opposite, which reveals that such an LLM is not suitable for a RAG system. In the domain of pharmacometrics, the ground truth provided by the retrieved context needs to be accurately extracted without any significant inconsistencies. Using faithfulness as a metric will help us gauge which model is better suited for AskPumas, as we require a model that is proper and grounded in the context provided, minimizing the risk of generating inaccurate or misleading information.

RAGAS can extract this score by utilizing an external LLM to generate multiple statements based on the answer provided by the LLM. Then, it proceeds to cross-verify if these statements are true based on the retrieved context. Finally, RAGAS calculates a score (see Eq. 1), where

the total number of statements generated is denoted $|S|$ and the total number of supported statements based on the context is denoted by $|V|$.

$$F = \frac{|V|}{|S|}. \tag{1}$$

2.6.2 Answer Relevancy

Answer relevancy measures how directly LLM's outputs address the questions from the prompt repository. If the answer is able to generate outputs that are on topic and directly respond to the prompt, then such an output will have a high answer relevancy score. This shows that an LLM that can achieve higher answer relevancy scores is less likely to hallucinate in a RAG system and can interpret questions clearly. To add on, lower scores in this metric indicate that such models are unable to provide answers that are relevant to the prompt posed to them.

This metric does not test for factuality or accuracy, but rather the quality of the LLM output with regards to redundant or missing information that does not address the question fully; we want to identify LLMs where the outputs are closely aligned with the question at hand. An LLM with a higher answer relevancy score will be able to answer questions that are complicated and much harder to interpret, for example, a question that uses a high amount of formal logic or a question that asks about drug modelling.

RAGAS first prompts an external LLM to generate n artificial prompts based on the answer provided by the LLM. Then, it calculates a score by using the cosine similarity between the vector embeddings of the n artificial questions and the original prompt (see Eq. 2) (Es et al. 2025).

$$\text{AR} = \frac{1}{n}\sum_{i=1}^{n} \sim \Big(e(q_i'),\, e(q)\Big), \tag{2}$$

where $e()$ is the embedding function, q is the original question, and q_i' are the artificial questions.

RAGAS takes in three inputs: the prompts posed to the LLM (prompt repository), the output of all 17 LLMs (generating LLM output stage), and the retrieved context provided to the RAG model for all prompts. Using these inputs, it automatically evaluates the answer relevancy and the faithfulness metric, providing us with quantitative scores. We interpret these scores based on the metrics, providing valuable insights into the quality of all our LLM outputs.

2.7 LLM Rubric – Grading Output with Assistance from Another LLM

The second evaluation method incorporates the use of an external LLM, specifically OpenAI's `gpt-5-mini`, to grade all generated outputs of our 17 LLMs. This provides an additional quantitative metric, which is a valuable secondary score in addition to RAGAS. To achieve this, we decided to create an extensive rubric, which is provided as context to the external LLM that grades the outputs . Using a rubric makes this method of evaluation standardized, as the external LLM can objectively grade generated outputs consistently (see Table 1). This objective method of evaluation is sensitive in nature as it penalizes models by assigning lower scores to outputs that are not up to a certain standard; for instance, the output lacks adequate depth, or the output is not relevant to the given prompt.

Table 1: Rubric used for LLM evaluation

Criteria	Benchmark Context	Band 1	Band 2	Band 3	Total Score
Relevance	Is the answer relevant to the question asked?	(17-20 points) The answer is directly relevant to the question asked. Addresses the main points and subpoints of the question. No off-topic information is included.	(10-16 points) The answer is mostly relevant to the question. Covers some main points but may miss some subpoints. Minor off-topic information may be included.	(0-9 points) The answer is not relevant or only partially relevant to the question. Misses most of the main points and subpoints. Includes significant off-topic information.	20
Comprehensiveness	Does the answer cover all aspects of the question completely?	(20-25 points) The answer covers all aspects of the question completely. Provides thorough and exhaustive coverage of the topic. Includes all necessary information and details.	(10-19 points) The answer covers the main aspects of the question but not completely. Provides adequate but not exhaustive coverage of the topic. Some necessary information or details may be missing.	(0-9 points) The answer covers few or none of the aspects of the question. Provides insufficient coverage of the topic. Many necessary information or details are missing.	25
Clarity and Coherence	Is the answer clearly written and easy to understand? Does it follow a logical structure?	(20-25 points) The answer is clearly written and easy to understand. Follows a logical structure with well-organized content. Free from grammatical and spelling errors.	(10-19 points) The answer is generally clear but may have minor issues affecting understanding. Follows a mostly logical structure with some organizational issues. Contains minor grammatical or spelling errors.	(0-9 points) The answer is unclear or difficult to understand. Lacks logical structure and organization. Contains significant grammatical or spelling errors.	25
Depth and Detail	Does the answer provide sufficient detail and depth? Does it address the nuances of the question?	(25-30 points) The answer provides sufficient detail and depth. Addresses the nuances of the question thoroughly. Uses examples, explanations, or elaborations.	(15-24 points) The answer provides some detail and depth but is not thorough. Addresses some nuances of the question but not all. Includes some examples and explanations but lacks elaboration.	(0-14 points) The answer lacks sufficient detail and depth. Does not address the nuances of the question. Lacks examples, explanations, and elaborations.	30

We also created 3 different bands of scores for each category, with each band having specific descriptions about the expected quality of the answer to attain that score. Creating these

bands allowed the external LLM to easily judge the generated output with a high level of granularity. For instance, consider the comprehensiveness category in our rubric (see Table 1). Here, if the generated output covers all parts of the question, then the external LLM will assign a grade according to Band 1. If the generated output misses some parts of the question, then the external LLM will assign a grade according to Band 2. Creating these microinstructions for the LLM from a detailed bottom-up design significantly helps the LLM understand how to grade outputs according to our rubric criteria. This method leveraged the power of the LLM to sort through and objectively grade around ~3000 outputs.

2.8 Answer Rejection on Unanswerable Prompts

As described in Section 2.3, a subset of 14 prompts in the prompt repository was intentionally designed to be unanswerable, given the retrieved context. For these prompts, the desired behaviour is for the model to acknowledge that the retrieved context provided does not have the answer, rather than hallucinating and attempting to answer a prompt that is unanswerable.

We therefore evaluated these prompts on the model's ability of "answer rejection". For all 17 LLM's outputs on the 14 prompts, we manually checked the outputs and assigned:

- a score of 1 if the model indicated that it could not answer the prompt from the given context and linked the user to PumasAI discourses, and
- a score of 0 if the model attempted to answer the prompt, without indicating that it could not answer the prompt from the given context.

These scores override all three evaluation metrics: faithfulness, answer relevancy and the LLM rubric. For these 14 prompts in these three evaluation metrics, we replace them with binary answer rejection scores of 1 or 0, based on their behaviour. This penalises models that are unable to follow instructions clearly and reason critically about the incorrect evidence provided.

2.9 Clustering All Generated Outputs

After our two evaluation methods and generating three quantitative scores of Faithfulness, Answer Relevancy, and LLM rubric, we were able to compare model outputs using a fixed prompt repository. However, we wanted to go beyond our prompt-based evaluation - to deepen our analysis and gain more valuable insights. We wanted to directly examine the relationship between all LLM outputs amongst each other. Hence, we utilized the k-medoids clustering algorithm – a method to see the similarity and differences between LLM outputs across all LLMs.

First, we converted all 181 outputs of each LLM in our 17 LLM shortlist to vector embeddings. This transformation enabled us to represent text outputs (string format) as vectors, which

is required for the k-medoids clustering algorithm. We converted each output to a vector embedding using OpenAI's `text-embedding-3-small` model. Then, we used these embeddings to compute a 17×17 pairwise distance matrix D (see Eq. 3), based on the cosine distance between our vectors.

Each entry in matrix D is the average cosine distance value between the i-th and j-th vector embeddings v, where i and j represent two different LLMs, for all $k = 1, 2, 3, \ldots, 181$ representing each prompt. Here, the `cosine_distance` function is calculated as $1-$cosine similarity between two vectors of different LLMs for the same prompt. Finally, the pairwise distance matrix D was our input for the k-medoids clustering algorithm.

$$D_{ij} = \frac{1}{K} \sum_{k=1}^{K} \text{cosine\_distance}\Big(v_{i,k},\, v_{j,k}\Big), \tag{3}$$

where $K = 181$, which is the number of prompts.

3 Results

After generating the outputs of all 17 LLMs, we used numerous Python scripts to run each evaluation method (faithfulness, answer relevancy, and the LLM rubric) to produce quantitative metrics for 181 outputs. We then took the average score of all 181 outputs to obtain a single representative score for each LLM. Further, after normalizing the scores, we calculated the total score, which is simply the average of all three metrics (see Table 2). Our rankings are presented in descending order based on the LLM's total score.

Table 2: Evaluation Results

Cluster	Model Name	Faithfulness	Answer Relevancy	LLM Rubric	Average Score
G	openai/gpt-5-chat	0.810584	0.88392	0.877735	0.857413
Y	google/gemini-2.5-flash-lite	0.891469	0.841338	0.769558	0.834122
G	google/gemini-2.5-pro	0.828043	0.854472	0.809779	0.830765
Y	google/gemini-2.5-flash	0.942048	0.763844	0.766464	0.824119
Y	x-ai/grok-4-fast	0.898793	0.749754	0.81558	0.821376
G	qwen/qwen3-max	0.719511	0.827287	0.831602	0.7928
G	mistralai/mistral-small-3.2-24b-instruct	0.757549	0.862015	0.740387	0.78665
G	mistralai/mistral-medium-3.1	0.646809	0.859995	0.848674	0.785159
Y	deepseek/deepseek-chat-v3.1	0.739071	0.818818	0.776022	0.777971
Y	anthropic/claude-sonnet-4.5	0.710129	0.783852	0.839227	0.777736
G	openai/gpt-oss-20b:free	0.560521	0.872315	0.854586	0.762474
Y	anthropic/claude-haiku-4.5	0.810224	0.710443	0.744696	0.755121
G	anthropic/claude-opus-4.1	0.601006	0.819684	0.829006	0.749899

Table 2: Evaluation Results

Cluster	Model Name	Faithfulness	Answer Relevancy	LLM Rubric	Average Score
Y	google/gemma-3-12b-it	0.763617	0.769941	0.710608	0.748055
G	openai/gpt-5-nano	0.786692	0.589426	0.857735	0.744617
G	z-ai/glm-4.6	0.700476	0.773206	0.71895	0.730877
B	openai/gpt-5-mini	0.747569	0.520762	0.879116	0.715816

After converting the LLM outputs to vector embeddings and creating the pairwise distance matrix D, we ran the k-medoids clustering algorithm to cluster LLM outputs (see Section 2.9). We tested the algorithm by regenerating vector embeddings and reclustering them three times, achieving consistent results that demonstrate the robustness of our clustering solution. From the algorithm, we determined 3 clusters, as represented in our final rankings (see Table 2), where:

- Cluster G which represents the color green.
- Cluster Y which represents the color yellow.
- Cluster B which represents the color Blue.

To add on, we plotted our clustering results along with our evaluation methods (faithfulness, answer relevancy, and LLM rubric) to observe any noticeable trends and behaviors of these LLMs:

- The first plot displays the faithfulness score of an LLM against the answer relevancy score of the same LLM, colored by the cluster representation (see Figure 1).

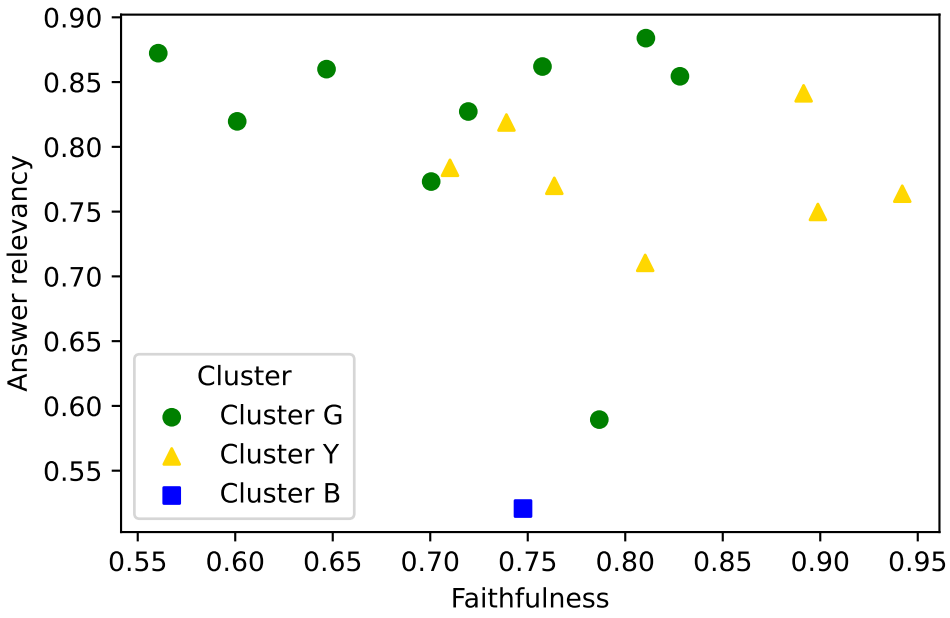

Figure 1: Faithfulness vs answer relevancy for all LLMs, colored by cluster.

- The second plot displays the LLM rubric score of an LLM against the answer relevancy score of the same LLM, colored by the cluster representation (see Figure 2).

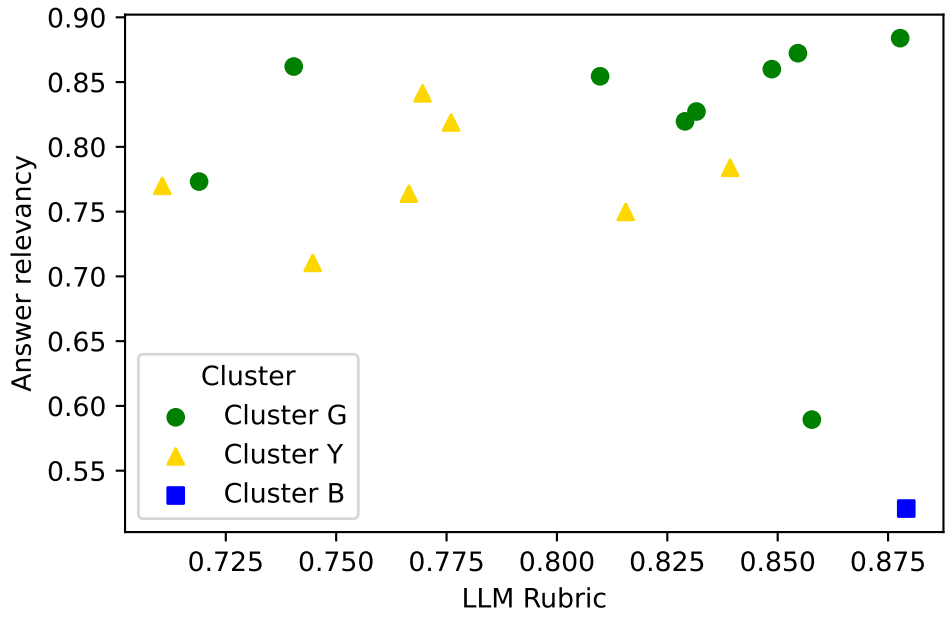

Figure 2: LLM Rubric vs answer relevancy for all LLMs, colored by cluster.

Finally, we created a table to summarize the clustering results to report, for each cluster, the mean of the models' average scores. This helps us understand how well the clusters seperate models by our evaluation metrics.

- This table displays mean average scores of the three clusters, colored by cluster representation (see Figure 3).

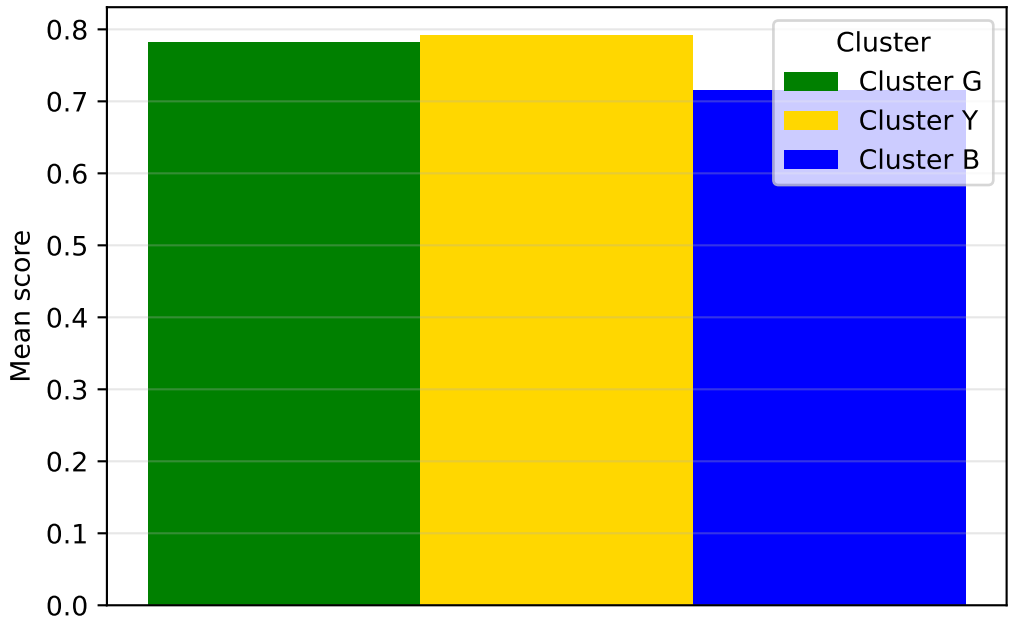

Figure 3: Mean metric values by cluster.

4 Conclusion

We observe that the top 5 LLMs for a RAG system for AskPumas, with respect to use cases involving Pumas, Julia, and pharmacometrics, are `gpt-5-chat`, `gemini` models, `qwen3-max`, and `grok-4-fast` (see Table 2).

To obtain an overall ranking, we aggregated the three metrics (faithfulness, answer relevancy, and LLM rubric) using an unweighted average (see Section 3). This is to treat all three dimensions as equally important, which is the primary purpose of choosing these metrics. However, it is important to note that changing these weights would significantly alter the rankings.

For instance, `gemini-2.5-flash` outperforms `gpt-5-chat` by 13% in faithfulness, though `gpt-5-chat` has a better average score as it scored better in other metrics (see Table 2). If faithfulness has more weight, workflows that prefer to emphasize faithfulness in their RAG application would see `gemini-2.5-flash` as a better option than `gpt-5-chat`. This implies that the best model is relative and highly dependent on the use case of the LLM. Having a dynamic evaluation method allows us to consider different metrics, ensuring that model selection for AskPumas best aligns with PumasAI's priorities and goals. This helps in making a more informed decision, rather than relying on a singular metric.

The results show that the difference between each average score is very low. Consider the average scores for `gemini-2.5-flash-lite`, `gemini-2.5-pro`, `gemini-2.5-flash`, and `grok-4-fast`, which are in the top 5 LLMs according to the rankings (see Table 2). Scores here have a very small percentage difference of $\sim 1\%$ or 2% between consecutive models in our rankings. These differences are minimal; here, we cannot definitively assert that one model is "better" than the next consecutive model in the rankings. All of the top-performing models can be treated in the same tier as compared to lower-scoring models in the rankings of our evaluation metrics. Our table of mean average scores reveals the same, where both Cluster Y and Cluster B have similar mean average scores (see Figure 3). This means we have two styles of strong models, and we should not overinterpret minor percentage differences.

Some models can be self-hosted. Running LLMs locally is crucial for data privacy and is an imperative consideration for AskPumas. For example, `deepseek-chat-v3.1` and `mistralai/mistral-small-3.2-24b-instruct` have the highest scores amongst the models that can be self-hosted and can be considered as viable candidates for AskPumas. The choice between a self-hosted and API-based LLM is ultimately dependent on the sensitivity of your data for an AI application. Developers should weigh the trade-offs, such as maintenance, cost, latency, and scalability, which are often overlooked when self-hosting an LLM, and select a suitable model for their application requirements.

When running the clustering algorithm, we decided to choose the results showcasing three clusters $k = 3$, where k is the number of clusters. This choice was guided by the fact that varying the number of clusters revealed different information, based on the metrics:

- The results for $k = 2$ and $k = 3$ are almost the same, the only difference being that $k = 3$ had assigned `gpt-5-mini` to its own cluster (Cluster B). We observe that `gpt-5-mini` is indeed an outlier. It is ranked last and has the lowest average score in our evaluation results (see Table 2). Notably, it is the highest-scoring LLM in the LLM rubric metric, although this is expected given that `gpt-5-mini` was used in grading the model for LLM rubric, and therefore would tend to favour its own outputs (see Section 2.7).
- When $k >= 4$, there were too many cluster groups, which did not provide any relevant insights about the models. Hence, we did not choose four or more clusters in our final evaluation results.

In addition, the clustering results plotted along with our evaluation methods (faithfulness, answer relevancy, and LLM rubric) provide valuable insights:

- From the faithfulness vs. answer relevancy plot (see Figure 1), we observe that Cluster Y models have significantly higher faithfulness scores than Cluster G models. Although Cluster Y has slightly lower answer relevancy scores than Cluster G.
- From the LLM rubric vs. answer relevancy plot (see Figure 2), we observe that Cluster G models have significantly higher LLM rubric scores than Cluster Y models. We once again observe that Cluster Y has slightly lower answer relevancy scores as compared to Cluster G.

From these plots, we observe that the Cluster G models have higher answer relevancy and LLM rubric scores but lower faithfulness than yellow (see Figure 1, Figure 2). These models are willing to give on-topic answers, but they are prone to not interpret and apply the provided context in their outputs. A representative model that shows this behavior in our rubric is `mistral-medium-3.1` - it has a high answer relevancy score and LLM rubric score, but one of the lowest faithfulness scores (see Table 2).

Alternatively, we observe that Cluster Y models have higher faithfulness scores but somewhat lower answer relevancy and LLM rubric scores (see Figure 1, Figure 2). These models stay closer to the context and are more conservative. This implies that sometimes, they are not answering the question fully or are not direct. A representative model that shows this behavior in our evaluation is `gemini-2.5-flash` - it has the highest faithfulness score out of any other model but does not have relatively high answer relevancy or LLM rubric scores, thus reducing its average score (see Table 2).

The code for this evaluation framework is available at https://github.com/PumasAI-Labs/AskPumas-LLM-Evaluation-Paper.

5 Discussion

5.1 Importance and Use Case

More than the specific results of our evaluation framework, the methodology presented in this paper serves as a guide for scientists to select a suitable LLM; it certainly helped PumasAI confidently choose an LLM for AskPumas. With the constantly changing landscape of high-performing LLMs, we now have a structured method to consistently evaluate LLMs for a RAG application. Understanding the nuances presented in the paper, including factors to choose the correct LLM, data privacy and compliance, and our evaluation methodology, will help make a more informed decision in a systematic and context-aware approach.

The framework is a modular process where scientists can adopt different parts of the framework to their specific RAG architecture. For instance, in use cases of law and medicine where hallucination is unacceptable, prioritizing answer rejection is crucial (see Section 2.8). We observed that the performance gap between subsequent LLMs in our evaluation is only ~1-2% (see Table 2). This suggests that for domain-specific RAG applications, operational factors such as cost, implementation, and model size outweigh raw benchmark scores (see Section 2.1).

Another advantage of our modular approach to evaluation, allows scientists to prioritize LLM selection based on their use cases. Whether selecting an LLM purely by domain-specific average evaluation scores or prioritizing one metric over another, the flexibility of our framework shows that there are many styles of strong models (see Section 4). This is important for scientists building a RAG architecture, as our research provides an understanding of which metric to prioritize, based on the use case of the application.

5.2 Limitations and Future Work

While our evaluation method is useful for LLMs integrated into RAG applications, it is important to note that novel methods to retrieve information are being developed. New methods, such as Corrective Retrieval Augmented Generation, can significantly improve the performance of RAG architectures by utilizing a method to assess the overall quality of retrieved documents (Yan et al. 2024). Seeing significant upgrades to RAG architectures, including the introduction of autonomous AI agents into RAG pipelines through Agentic RAG (Singh et al. 2025), our methodology is not tailored to evaluate LLMs in these refined RAG-based architectures. These frameworks will require a different approach than our proposed framework to individually evaluate LLMs and assess them based on the context and use case.

With AskPumas's RAG architecture evolving over time and upgrading to these more sophisticated, agent-driven architectures, this provides an opportunity for future work to refine our evaluation methodology to assess these upgraded domain-specific RAG architectures. By leveraging our current evaluation methodology, we can create a new framework and apply it to upgraded RAG architectures. Relevant methods in our current methodology for future RAG evaluation frameworks include exploring LLMs (see Section 2.1), creating a domain-specific prompt repository (see Section 2.3), and clustering LLMs using the k-medoids algorithm (see Section 2.9). By building on our current methodology, we can develop robust evaluation metrics that account for refined RAG architectures such as CRAG and Agentic RAG.

6 References

Chen, Jiawei, Hongyu Lin, Xianpei Han, and Le Sun. 2023. "Benchmarking Large Language Models in Retrieval-Augmented Generation." https://arxiv.org/abs/2309.01431.

Es, Shahul, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. 2025. "Ragas: Automated Evaluation of Retrieval Augmented Generation." https://arxiv.org/abs/2309.15217.

Hugging Face, Inc. n.d. "Hugging Face Hub." https://huggingface.co/docs/hub/en/index.

Inc., Ollama. n.d. "Ollama Documentation." https://docs.ollama.com/.

Korsbo, Niklas, Mohamed Tarek, Chris Elrod, Antoine Soubret, Francesco Brizzi, Christopher Rackauckas, Jogarao Gobburu, and Vijay Ivaturi. 2023. "DeepPumas for Automatic Discovery of Individualizable Functions Governing Longitudinal Patient Outcomes." In *PAGE 31 (2023) Abstr 10522*. https://www.page-meeting.org/?abstract=10522.

LangChain. n.d. "LangChain GitHub Repository." https://github.com/langchain-ai/langchain.

Lewis, Patrick, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, et al. 2021. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." https://arxiv.org/abs/2005.11401.

Liang, Percy, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, et al. 2023. "Holistic Evaluation of Language Models." *Transactions on Machine Learning Research*. https://openreview.net/forum?id=iO4LZibEqW.

Murtiyoso, Murtiyoso, Imam Tahyudin, and Berlilana Berlilana. 2025. "A Systematic Review of Retrieval-Augmented Generation for Enhancing Domain-Specific Knowledge in Large Language Models." *Sinkron* 9 (June): 969–77. https://doi.org/10.33395/sinkron.v9i2.14824.

Rackauckas, Chris, Yingbo Ma, Andreas Noack, Vaibhav Dixit, Patrick Kofod Mogensen, Simon Byrne, Shubham Maddhashiya, et al. 2020. "Accelerated Predictive Healthcare Analytics with Pumas, a High Performance Pharmaceutical Modeling and Simulation Platform."

Shanmugarasa, Yashothara, Ming Ding, Chamikara Mahawaga Arachchige, and Thierry Rakotoarivelo. 2025. "SoK: The Privacy Paradox of Large Language Models: Advancements, Privacy Risks, and Mitigation." In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, 425–41. ASIA CCS '25. ACM. https://doi.org/10.1145/3708821.3733888.

Singh, Aditi, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. "Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG." https://arxiv.org/abs/2501.09136.

Wang, Yubo, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, et al. 2024. "MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark." https://arxiv.org/abs/2406.01574.

Yan, Shi-Qi, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. "Corrective Retrieval Augmented Generation." https://arxiv.org/abs/2401.15884.

Yang, Xiao, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, et al. 2024. "CRAG – Comprehensive RAG Benchmark." https://arxiv.org/abs/2406.04744.

A Selected and Eliminated LLMs

Below, we list all models selected and eliminated for evaluation during the human vetting step (see Section 2.4), along with their implementation method.

Selected for Evaluation	Implementation
openai/gpt-5-chat	API
google/gemini-2.5-flash-lite	API
google/gemini-2.5-pro	API
google/gemini-2.5-flash	API
x-ai/grok-4-fast	API
qwen/qwen3-max	API
mistralai/mistral-small-3.2-24b-instruct	Open weights and API
mistralai/mistral-medium-3.1	API
deepseek/deepseek-chat-v3.1	Open weights and API
anthropic/claude-sonnet-4.5	API
openai/gpt-oss-20b:free	Open weights and API
anthropic/claude-haiku-4.5	API
anthropic/claude-opus-4.1	API
google/gemma-3-12b-it	Open weights and API

Selected for Evaluation	Implementation
openai/gpt-5-nano	API
z-ai/glm-4.6	Open weights and API
openai/gpt-5-mini	API

LLMs Eliminated	Implementation
microsoft/phi-4	Open weights and API
meta-llama/llama-3.3-8b-instruct	Open weights and API