ASSESSING LLMs FOR NATURAL LANGUAGE SEARCH IN POETRY DATABASES

by

RYAN KOVATCH

A THESIS

Presented to the Department of Computer Science
and the Robert D. Clark Honors College
in partial fulfillment of the requirements for the degree of
Bachelor of Science

May 2025

An Abstract of the Thesis of

Ryan Kovatch for the degree of Bachelor of Science
in the Department of Computer Science to be taken June 2025

Title: Assessing LLMs for Natural Language Search in Poetry Databases

Approved: *Eric Wills, Ph.D.*
Primary Thesis Advisor

This project aims to evaluate the performance of two major large language models (LLMs) — Google's Gemini and Meta's Llama — on their abilities to identify poems in a database based on abstract characteristics of the text, namely form and techniques, major themes, and historical context. Currently, search engines for poetry databases are primarily keyword-based, which makes the discovery of new poems difficult when the user does not know (or cannot predict) the content of a poem they are looking for. This paper uses a dataset developed by Walsh et al. (2024) and a technique called retrieval-augmented generation (RAG) to implement a natural-language search function where users can ask more advanced questions of a poetry collection and retrieve relevant results to abstract queries. On a benchmark, the LLMs performed poorly at poetry retrieval tasks across the board, scoring low on precision and recall, but patterns in the data indicated that the models performed better on some types of queries than others, and that the technology could become feasible for this use case sometime in the near future.

Acknowledgements

To my committee — Dr. Eric Wills, Dr. Rachel Elizabeth Weissler, and Dr. Barbara Mossberg — thank you for fostering the precise intersection of study that allowed this thesis to happen. Even at my least productive, you were all steadfast in supporting me. Your guidance throughout this process was invaluable.

I'd like to thank my various mentors in poetry, especially at the University of Oregon: Clara Trippe, Betsy Bonner, Cait Phenix, and Matthew Dickman. You all stoked a passion for poetry in me again and again. Thank you for keeping me in touch with the art form and with my fellow poets. Speaking of... to my Kidd Workshops cohort: Maya, Sophia, Anna Liv, Danielle, Tristan, Angel, Mateo, Roya, Lola, and Asia. You are the reason I write. Getting to spend a whole year honing my craft with you all was the single greatest privilege of my undergrad; if I could relive it, I would again and again.

To my dear friends, who let me be annoying about this project all the time: I cherish you all. This list could never be exhaustive (we'd run out of pages), but some of you really earned a shoutout. To the greatest trivia team of all time, Gaysexy — Jess, Ally, Marina — and to our extended queer family: Breighton, Morgan, Grace, Lia, Ayesha. Everything I do is to make it back to you all. To the Girl More Girls, who have been there with me since the beginning: Kristin, Kennedy, Sophia, Mollie, Riley, Allie, and Natalie. All of my poems about friendship have been about y'all. And to my friends who live far from me now but are never more than a phone call away: Leah, Harper, and Hafsa. If I can keep up, we'll get right back to it.

Table of Contents

List of Tables

Introduction

Since the introduction of large language models (LLMs) into popular culture, poetry has commonly been used as a demonstration of productive language capabilities. It is a uniquely multimodal and multifaceted form of language use, encompassing "many challenging aspects of human language and expression," including "verbal, aural, and visual elements" which may incorporate "deep emotion and meaning in non-literal, ambiguous ways" (Walsh et al. 2). To any speaker, a good poem can represent a kind of mastery of language production — a measured application of a language's elements to convey something nuanced and particular. This is a concept that marketers for LLM products have latched onto.

In an effort to show that LLMs "understand" language very well, tech companies have gone out of their ways to associate them with poetry. As Walsh et al. pointed out, some of Anthropic's Claude models are codenamed after poetic forms. The LLM integrated into Microsoft's Bing, now called Copilot, has advertisements encouraging users to ask it for poems and even company-authored how-to guides written on generating poetry. And Google's Gemini, originally named Bard (another word for "poet"; Walsh et al. 1), has multiple help documents where writing poetry is presented as a use case akin to drafting a letter or email ("Gemini Web App"; "Write with Gemini").

LLMs' actual comprehension of poetry is difficult to ascertain, however. Generated poems are produced "not through human thought, but through the analysis of massive amounts of linguistic data," raising the question of whether they possess any meaning at all (Binder 356). The process of generating poems itself is challenging and imperfect "due to the complex interplay of style, meaning, and human emotion" present, and the resulting poetry "lacks diversity" (Zhang and Eger 1-2). Much less research has been done on models' *receptive*

understanding of poetry — with large language models being particularly absent due to their proprietary, closed-source nature and costs associated. The primary research this thesis builds upon is a paper by Walsh et al., "Sonnet or Not, Bot? Poetry Evaluation for Large Models and Datasets," which builds a database of poetry and develops a new methodology for testing LLMs' abilities to discern between various forms and formal aspects in poetry. They found that LLMs "can successfully identify both common and uncommon fixed poetic forms [...] at surprisingly high accuracy levels," but struggle with "unfixed" poetic forms that are "based on topic or visual features" (Walsh et al. 2).

Walsh et al. only tested the models' performance when the poem was explicitly given, however. This thesis intends to expand upon their research by developing a retrieval-augmented generation (RAG) task that requires models to identify poems with certain attributes (including form) from a *collection* of poems. This will require the models to consider each poem in the collection and return a list of poems matching certain criteria, rather than act as an isolated poetic form discriminator.

If any model performs sufficiently well at this task, it can behave as a more capable alternative to search algorithms currently used in prominent poetry databases, such as those of the Poetry Foundation or the Academy of American Poets. The present search functions of these websites only allow keyword-based search, which requires the user to know (or be able to predict) the verbatim text content of a poem, which is not always feasible. For example, if a user is searching for epistolary poems (those written in the form of letters), what keyword could they search? These poems do not typically have the words "epistolary" or "epistle" in them, or even "letter." They may start with the word "dear," but plenty of poems that are not epistolary have this word. Same with common sign-offs like "sincerely." It is thus a needlessly difficult task to

find poems like this that one has not read before. The database maintainers solve this problem in part with human custodians who tag poems as having certain poetic forms (and these tags are critical to the present research), but this is an onerous task, especially when committing many new poems at once, and even human experts do not perform very accurately at it (Walsh et al. 6). A sufficiently competent LLM could relieve the burden of manual tagging and return accurate results for more complex queries, solving both problems at once.

This thesis aims to evaluate whether current LLMs are capable enough to retrieve poems from a database accurately based on formal, thematic, and historical contextual attributes, and whether this capability could reasonably replace current methods of poetry search. A model suitable for search functionality would return results with high precision and recall, low latency, and other metrics as detailed in the Methods section. Additionally, it aims to determine how performance differs on retrieval tasks as opposed to identification tasks. The project will use the dataset compiled by Walsh et al. and methods developed by Wu et al. to benchmark LLM performance on retrieval tasks to investigate this.

Literature Review

This project sits at the intersection of the nascent field of LLM-assisted poetry evaluation and the growing field of document parsing with retrieval-augmented generation (RAG).

RAG has proven very helpful in fields where large amounts of unstructured data exist. A 2023 project by Peng et al. used RAG to assist farmers with pest identification. Because "most farmers are not capable of accurately identifying pests in the field," they usually consult digital sources for information, but "there is a limited number of structured data sources available" for this purpose, so it is difficult to identify pests quickly (Peng et al. 1). Their solution was to supply a corpus of pest information documents to an LLM, which, over multiple passes, produced a structured and easily-searchable map of pests and their attributes (Peng et al. 2-4). This makes pest data much more accessible to farmers by making searching for pest attributes easier.

RAG is also common in implementing chat bots that draw from information in specialized datasets. This has become especially prominent in clinical fields, whose knowledge bases are comprised of thousands of text documents containing mostly unstructured information. A chat bot called LiVersa developed by Ge et al. was able to answer questions about liver disease and hepatology with high accuracy after ingesting just 30 guidance documents from the American Association for the Study of Liver Diseases. Even with such a small corpus, a panel of 15 experts rated LiVersa more accurate than ChatGPT, demonstrating how effective RAG is at specializing a language model. Still, its small corpus meant that it produced less comprehensive answers and more hallucinations than ChatGPT — but these issues are largely fixed by increasing the number of documents ingested (Ge et al. 4).

These examples highlight why poetry databases may be a prime candidate for RAG. The poetry database is an inherently unstructured collection that requires significant human effort to structure, like the agricultural pest dataset, and we want to be able to answer more specialized questions than a keyword search can, like in the hepatology example. But is there an indication that LLMs can infer the attributes we want to search for from different poems? Scholarship on LLMs' ability to accurately evaluate poetry is slim.

"Sonnet or Not, Bot? Poetry Evaluation for Large Models and Datasets" by Walsh et al. is one of very few studies to examine LLM performance on poetry categorization tasks. (Most other studies on receptive poetry comprehension use smaller or special-purpose language models that do not reflect the capabilities of LLMs). Using poems from the Poetry Foundation and Academy of American Poets, as well as a few manually digitized poetry books, they build a corpus of over 4,100 poems with metadata about authorship (including birth and death dates), form, publication date, overarching themes, and occasion (i.e. holidays or times of year pertaining to the poems; Walsh et al. 3-4). They then test six major LLMs using four different prompts that revealed varying amounts of a poem to the LLMs. These prompts ask the LLMs to categorize the poem based on a provided list of forms (sonnet, ballad, villanelle, etc.) and provide a brief rationale for their decision as well as a confidence rating (Walsh et al. 6).

Their findings indicate that the LLMs are better at categorizing poems with "fixed" forms, defined as following "particular patterns in terms of number of lines, meter, rhyme, and/or repetition," than "unfixed" forms, "defined by particular subject matter or kinds of content, rather than by patterns of repetition or sound." Additionally, they find that the models struggle with forms based on "abstract ideas and styles" like odes and "visual features" like in concrete poetry. These results may predict similar performance on RAG tasks involving the same forms.

At the end of their study, they release a subset of their corpus containing only public-domain poems, which will be the dataset my project uses (Walsh et al. 4). The Walsh et al. paper also provides a lot of important vocabulary for my thesis, such as the difference between “fixed” and “unfixed” forms, which will be important when examining the performance data I collect on the LLMs.

Methods

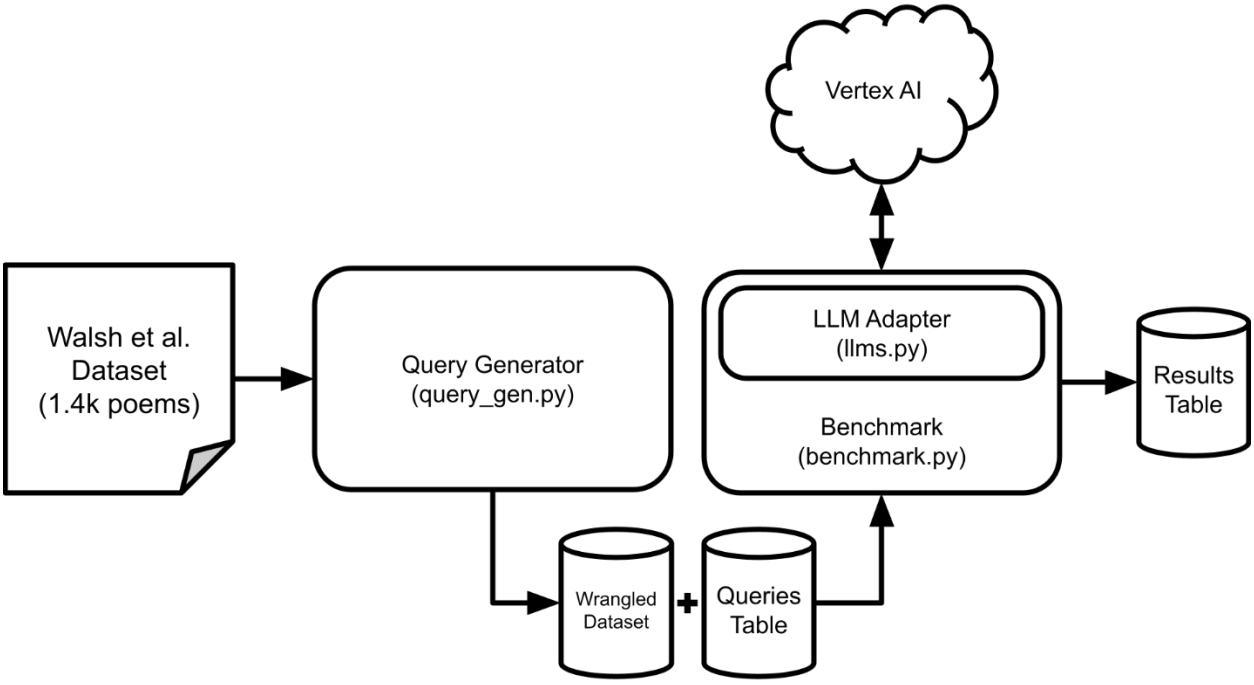

Figure 1: Diagram of components and their data flows in the benchmarking process.

The basic trajectory of the project is this: start with the dataset of poems compiled by Walsh et al. that have already been tagged manually (for example, as “Sonnet” or “Haiku”) by human reviewers, then create natural-language search queries based on these tags (such as “poems in sonnet form”) and see if the LLMs can identify the correct poems from a dataset without tags. This required several steps organized into a complex system.

Data Wrangling

Before I could upload the dataset for use with the LLMs, I needed to do some basic data wrangling to adapt it for my purposes. The code that performs these tasks accounts for the first half of the query generator (query_gen.py), as marked by code comments.

I began by assigning each poem an alphanumeric ID of the form ‘P’ + a number ≥ 777,000. An example is P777675, which corresponds to “Dejection: An Ode” by Samuel Taylor Coleridge. I chose to start the IDs with P77 to discourage the LLM from hallucinating poem IDs

and making it more obvious when such hallucination occurs. This was helpful during the development of the system prompt, as early iterations would occasionally have the model spit out invalid IDs such as P000001. Because identifiers of this format are uncommon, when the model produces them, we can be more certain that they are being lifted from the poetry dataset as opposed to hallucinated/fabricated based on training data or instructions in the prompt.

Then I dropped a set of irrelevant columns from the dataset, namely: "author_link", "collected_from", "form_tags", "theme_tags", and "occasion_tags." The first two were dropped as they contained information that couldn't be used to classify the poems. The latter three columns did contain a useful organization of tags into form, theme, and occasion types, but these categories were not entirely useful to this research. All tags listed in the latter three columns were also present in the "tags" column, so they could be dropped without data loss.

After this, I stripped out tags that were irrelevant to the form or content of the poems. These were usually related to the poems' organization or placement on the websites they were scraped from. They were:

> "Appeared in Poetry Magazine," "Audio," "Learning Resource," "Public Domain," "Related Audio," "Related Video," "Spanish," "Suitable for Children," "Suitable for Teens," and "Translation."

The following tags were also dropped: "Imagist," "Landscapes & Pastorals," "Mixed," "Other Religions," "Symbolist," and "Verse Forms." These tags either had too few poems (less than ten) or had meanings that were overly broad or confusing.

Some tags needed to be renamed to fit naturally in a search query. For example, the tag for poems written in couplets was singular ("Couplet"), but a person would almost certainly search for "couplets" instead. Additionally, some tags were duplicated with slightly different names, such as "Ekphrasis" and "Ekphrastic." The full list of replacements is defined as TAG_REPLACEMENTS in the query_gen_config.py file.

Finally, the “pub_year” column was useful, but rather sparse. The “poem_source” column often contained copyright years which could be used to supplement it. I wrote a loop to scan each poem’s source data for a copyright year. If one was found, then (1) it was recorded as the pub_year if there was not one already, or (2) if there was a pub_year already listed, the new year was recorded only if it was earlier. This helped fill in publication context for when we’d ask about certain time periods later (ex. “poems published during the European Renaissance”).

Query Generation

Once all of the data was wrangled, it could be used to generate search queries for benchmarking. The query generation program creates a new table with five columns:

1. “query,” the text of the search query;
2. “query_type,” the attribute of the poems that’s being referenced, such as “form”;
3. “operative_term,” the tag that the search results should capture;
4. “fixed,” a boolean column indicating whether a form or technique being searched is fixed or not;
5. and “golden_answers,” a list of poem IDs considered “correct” search results.

There are five types of queries being benchmarked. The first two, “form” and “technique,” can either be “fixed” or “unfixed.” The definitions of (un)fixedness are taken from Walsh et al., who originally ascribed them only to form. I extend them to techniques. A technique is similar to a “formal element” as defined by Walsh et al. (3), but expanded to include rhetorical choices that may be reflected in either the form or content of a poem. A fixed technique, such as alliteration, follows a particular pattern that is readily identifiable. An unfixed technique, such as allusion, is less rigidly defined and requires an understanding of the poem’s content to identify. The query generator program has a configuration file (query_gen_config.py) that defines which forms and techniques are fixed or unfixed.

Tags not listed as either forms or techniques are taken to be “theme” tags. Theme tags, like “Romantic Love,” generate theme queries, like “poems written about romantic love.” Themes, along with the remainder of the query types, have no fixedness. They strictly pertain to the content of a poem, and roughly describe what the poem is “about.”

The last two query types have to do with historical context surrounding a poem’s creation and publication. “historical_pub_context” queries ask the model for poems published during a certain time period, such as the American civil war. Likewise, “historical_author_context” queries ask the model for poems published by authors who lived during these historical periods. These queries require the model to examine a poem’s publication year and the birth and death dates of its author to determine whether it falls within a certain historical context. The historical periods under test are also defined in the query generator’s configuration file.

Form, technique, and theme queries are generated first. The program iterates over every tag in the Walsh dataset and checks if it has at least ten poems. If it does not, the tag is discarded. If it does, the dataset is filtered down to poems with this tag and their IDs are recorded as the golden_answers. Then, the tag is made lowercase and inserted into a number of predefined search query templates, such as “poems in {} form” or “poems on {},” each producing a final search query that is ready to be fed to an LLM. Each form and theme tag produces four queries, technique tags produce two, and historical periods produce six (three for publication year and three for author lifespan). These templates are defined in the configuration file. Forms and techniques are then marked with their fixedness.

For historical context queries, the program iterates over the historical periods provided in the configuration file. It then filters down poems (1) that were published during each period and (2) whose authors’ lives overlapped each period. The former group becomes the golden_answers

for a new historical_pub_context query (like “poems written during the first world war”), and the latter becomes the golden_answers for a new historical_author_context query (like “authors from the great depression”). Like before, if fewer than ten poems exist for a certain historical context query, it is excluded from the final query set.

Once execution completes, a “queries.csv” file is written to the working directory with the queries table, which came out to 510 rows.

LLM Integration

Now that I had a list of queries to ask the models, I had to actually integrate my code with the models themselves. Owing to a lack of compute resources and the fact that one of the models chosen was closed-source, I had to carry out text generation in the cloud. I chose Google Cloud’s Vertex AI platform for this purpose, as it offers a number of major language models as managed APIs. I simply had to click “Enable” in the cloud console to gain access to them.

I used an object-oriented model to create LLM interfaces that would be used in the benchmark. This began with an abstract “LLM” class from which all LLM interfaces would inherit. This class had only two methods: “embed_doc,” which would insert a provided document into the model’s vector database (the poetry corpus), and “generate,” which would generate a response given a query using the vector database as context. I then wrote a “VertexAIModel” class with a number of additional methods specific to Google Cloud, such as “batch_embed_from_gcs” which embeds a folder of files hosted on Google Cloud Storage. While this project only uses Vertex AI, separating out Google Cloud-specific functions like this allows someone else to implement LLM interfaces from another cloud platform if this research should expand in the future. After this abstract class was finished, two model-specific classes, “Gemini” and “Llama,” could inherit it and be made functional with only four lines.

The particular model versions chosen were Gemini Flash 2.0, the latest stable release of Google's high-performance Gemini model at the time of writing, and Llama 3.1 8B, the latest low-parameter model from Meta available as a managed API. The goal was to select the model with the smallest number of parameters (ideally less than 10B) from each family, as this model would use the least amount of compute and thus offer the lowest latency search capability, perhaps at the cost of performance.

Because these models are inherently less complex than their large-parameter counterparts, they are more sensitive to design decisions in the prompts given to them. This is a phenomenon that affects all LLMs; even small changes in phrasing or formatting can radically affect performance on a task (Cao et al. 1). I encountered this phenomenon many times while designing the benchmark. Once, a stray "otherwise" left behind in the prompt after editing out a clause resulted in Llama completely disregarding all instructions about how to format its responses. To try and curb the effect of this prompt sensitivity, I used a prompt engineering whitepaper published by Google for guidance on how to design the final prompt for the poetry retrieval. It resulted in this prompt, which was used for the final benchmark:

> Act as a poetry expert. Identify poems in the dataset that match the search query provided. Find as many poems as you can that match the query, returning no less than ten. Return only the poem IDs as a comma-delimited list, without file extensions. Order the poems by how closely they match the search query. Give no additional information when returning the list; only return the poem IDs.
>
> Search query: {}

This prompt reflects several of the principles in the "Best Practices" section of the whitepaper, including being concise, using verbs that describe the desired behavior ("act," "identify," "find," "return"), being specific about the output requirements and format, and using instructions over constraints (Boonstra and Google 54-57). It is specified in the llm_config.py file and held constant across all models and queries, only varying on the "{}," which is replaced

by the query text at runtime. With a constant system prompt designed for best results, we can better attribute variability in model performance to specific attributes of that model, not attributes of the provided data.

Benchmarking

The benchmark program (benchmark.py) uses the artifacts of all the previous stages to run a benchmark of over 500 queries on each model. It is rather simple: for each query in the queries table (written out after the query generation phase), it inserts the query into the system prompt, gives that complete prompt to the model, and processes the model's response. Because we ask for a comma-delimited list in the system prompt, we assume that the model will return one -- if it does not, or the model does not return *only* the comma-delimited list, we mark the response as malformed and cease processing it.

Otherwise, we loop over the list of returned poem IDs and look each one up in the dataset, checking to see if it has the tag we're looking for. During this process, we generate an "accuracy vector" which maps the poem IDs to their correctness: True for correct, False for incorrect, and None for invalid (in cases where the ID was hallucinated). This is useful for counting the number of results of each type and looking at their distribution. Then, we compute some standard performance metrics for information retrieval systems — precision, recall, F1 score — as well as the mean reciprocal rank (MRR) used by Wu et al. (7). We also record the latency of the response in seconds.

Each query generates a row in a results table where the query text, type, operative term, fixedness, model response text, malformedness, accuracy vector, precision, recall, F1, MRR, and latency are recorded. This table is written to disk as a CSV upon exit.

Results

Overall Model Performance

Query Type	Avg. Precision	Avg. Recall	Avg. F_1	Avg. MRR	Avg. Latency
Form	31.73%	5.18%	7.26%	0.6166666667	1.283478231s
Technique	20.48%	1.71%	3.00%	0.4541666667	1.217232677s
Theme	18.88%	3.68%	5.40%	0.5196564885	1.319795606s
Historical Author Context	55.16%	1.10%	2.14%	0.768115942	1.214069906s
Historical Pub Context	7.48%	0.22%	0.42%	0.225	1.250194361s
Grand Total	**21.48%**	**3.47%**	**5.12%**	**0.5366615067**	**1.303928905s**

Table 1: Average Gemini performance metrics by query type.

Query Type	Avg. precision	Avg. recall	Avg. F_1	Avg. MRR	Avg. latency
Form	20.32%	2.80%	4.26%	0.6964285714	5.059030879
Technique	16.20%	1.64%	2.85%	0.6458333333	1.346398929
Theme	18.28%	3.37%	5.00%	0.5585536119	3.524293437
Historical Author Context	43.87%	0.92%	1.79%	0.8018353175	1.322679011
Historical Pub Context	6.62%	0.18%	0.34%	0.5416666667	7.40976983
Grand Total	**19.68%**	**3.02%**	**4.54%**	**0.5902985268**	**3.569322242**

Table 2: Average Llama performance metrics by query type.

Performance on the poem retrieval task was poorer than expected across most areas. Gemini showed an average precision of just 21.48%, meaning around 2 in 10 returned poems were actually relevant to a given query, and Llama had similar performance at 19.68% average precision. However, the models showed promising trends in certain areas. The average MRR for both models was above 0.5, meaning that (in cases where the model *did* provide correct results) the first correct result was usually the first or second one returned, indicating that the models were able to rank by relevance to some degree. Additionally, almost two thirds of the queries (63.33%) returned at least one correct result. With regards to efficiency, the average latency for both models (excluding when Llama would time out; see “Malformed Responses”) was around

1.3 seconds, which is quite tolerable considering the amount of compute required and the added overhead of communication over the Internet.

The median number of results returned for both models was 9, indicating that the models stuck close to the minimum required number of 10 results as specified in the system prompt, but went under this amount fairly often. Unfortunately, this rendered the recall and F_1 metrics relatively useless as the models consistently returned a small number of results compared to the size of the dataset, which prevented them from capturing more than a fraction of the golden answers for each query.

By Query Type

Forms

	Gemini		Llama	
Form	Avg. precision	Std. precision	Avg. precision	Std. precision
Ars Poetica	0.00%	0	0.00%	0
Ballad	21.11%	0.1170628195	10.83%	0.09089178661
Concrete/Pattern*			0.00%	0
Dramatic Monologue	3.70%	0.06415002991	6.25%	0.125
Ekphrastic	22.22%	0	30.56%	0.04811252243
Elegy	69.44%	0.05555555556	69.44%	0.05555555556
Epic	38.14%	0.1178553057	38.84%	0.1055184399
Epistolary	0.00%		0.00%	0
Haiku*			0.00%	0
Ode	0.00%		0.00%	0
Pastoral	0.00%	0	0.00%	0
Persona	0.00%	0	0.00%	0
Prose	10.00%		5.63%	0.06574889099
Sonnet*	95.00%	0.1	95.00%	0.1
Grand Total	**31.73%**	**0.3396811574**	**20.32%**	**0.3107989412**

Table 3: Precision averages and standard deviations by form type and model.

Blank cells indicate values that could not be computed due to malformed responses (causing divide-by-zero errors). Forms marked with an asterisk (*) are considered fixed — the rest are unfixed.

Gemini's average precision for form queries was 31.73%, with 22.36% for unfixed forms and a staggering 95% for fixed forms, though this is only because (1) performance on sonnets was near 100% due to many sonnets having the word "sonnet" in the title, and (2) the model produced malformed responses for the other two fixed forms under test (haiku and concrete/pattern poems). Llama did not produce malformed responses for these forms, but instead produced all incorrect responses, so its average precision for fixed forms was reduced to 38% — more on par with its unfixed average of 15.79% and overall average of 20.32%.

As mentioned before, of the fixed forms, both models performed best at identifying sonnets due to the form's name appearing in poem titles. Neither model could identify haiku or

concrete/pattern poems from the dataset at all. Of the unfixed forms, both models identified elegy with the greatest average precision (69.44%), followed by epics, ekphrastic poems, and ballads. Prose poems and dramatic monologues were identified with the least precision across both models. The rest of the unfixed forms, including odes, epistolary poems, and persona poems, were never identified from the dataset.

Techniques

	Gemini		Llama	
Technique	Avg. precision	Std. precision	Avg. precision	Std. precision
Alliteration*	0.00%	0	0.00%	0
Allusion	11.81%	0.009820927516	22.22%	0.1571348403
Blank Verse*	0.00%	0	0.00%	0
Common Measure*			0.00%	
Couplets*	45.00%	0.07071067812	36.67%	0.04714045208
Free Verse	65.00%	0.4949747468	30.00%	0
Imagery	22.50%	0.03535533906	22.50%	0.03535533906
Metaphor	25.00%	0.07071067812	27.50%	0.03535533906
Quatrains*	0.00%	0	0.00%	0
Rhymed Stanza*	15.00%	0.07071067812	15.00%	0.07071067812
Grand Total	**20.48%**	**0.2491885989**	**16.20%**	**0.14548843**

Table 4: Precision averages and standard deviations by technique type and model.

Blank cells indicate values that could not be computed due to malformed responses (causing divide-by-zero errors). Techniques marked with an asterisk (*) are considered fixed — the rest are unfixed.

The average precisions for technique queries were 20.48% for Gemini and 16.2% for Llama. Surprisingly, both models performed better on unfixed techniques: Gemini showed 31.08% precision on unfixed techniques compared to just 12% on fixed ones, and Llama had a similar difference at 25.56% and 9.39%. Both models returned around the same number of responses on average regardless of whether the technique was fixed or not, indicating that this really was the result of a marked difference in performance, though much of Gemini's precision

can be attributed to the query "poems that use free verse," to which it returned three results that were all correct (100% precision).

Of the unfixed techniques, the models both performed best at retrieving free verse poems: 65% average precision for Gemini and 30% for Llama. For fixed techniques, the models performed best on poems with couplets, with Gemini scoring 45% average precision and Llama scoring 36.67%. These were followed in performance by metaphor, imagery, and allusion on both models (in that order). Quatrains, common measure, blank verse, and alliteration were unable to be identified from the dataset by either model.

Themes

Gemini			Llama		
Theme	Avg. precision	Std. precision	*Theme*	Avg. precision	Std. precision
Christianity	90.00%	0	Nature	100.00%	0
Nature	89.29%	0.0714285714	Christianity	89.72%	0.0055555556
Religion	82.50%	0.05	Religion	82.50%	0.05
Seas, Rivers, & Streams	72.92%	0.0416666667	Seas, Rivers, & Streams	72.92%	0.0416666667
Faith & Doubt	65.00%	0.0577350269	War & Conflict	68.54%	0.1302374967
War & Conflict	62.50%	0.05	Faith & Doubt	67.86%	0.0528120786
Living	60.00%	0.0816496581	Living	60.00%	0.0816496581

Table 5: The top seven themes by average precision for each model, with standard deviations.

The average precisions on theme queries for each model were only 0.5% apart — 18.88% for Gemini and 18.28% for Llama. The list of themes that yielded the best average precisions were very similar between models; the top eight themes were an identical set but appeared in slightly different order. They were (alphabetically): Christianity; Death; Faith & Doubt; Living; Nature; Religion; Seas, Rivers, & Streams; and War & Conflict. For Gemini, Christianity was the most precisely identified theme with an average precision of 90.00%, followed by Nature (89.29%) and Religion (82.5%). For Llama, Nature was the top theme at 100%, followed by

Christianity (89.72%) and Religion (82.5%). This is likely correlated with prominent themes existing in the literary training data for each model, though we can't be sure as this data (and the methods for gathering it) are confidential and proprietary.

Historical Context

Historical context queries were perhaps the most complex type in the benchmark. The models were required to first convert the name of a historical period to a range of years (and approximate this range in some cases) before comparing this range to a value in each poem's metadata. The content of the poem, which was relevant to all other query types, was irrelevant to these requests, though it could still be used to help place a poem depending on if the poem was *about* the events of that historical period. As a result, performance varied wildly on these queries.

Context Type		Author Context		Publication Context	
Period	Length	Gemini	Llama	Gemini	Llama
American civil rights movement	14	0.00%	71.11%		
American Civil War	4	63.33%	80.00%		22.50%
American colonial era	292	73.33%	37.04%	0.00%	
American Revolution	18	42.22%	36.67%		
European Renaissance	300	73.89%	56.67%		9.26%
First World War	4	68.52%	65.56%	0.00%	0.00%
Great Depression	10		65.48%		0.00%
Industrial age	210	73.33%	7.87%	7.50%	
Machine age	65	53.33%	3.70%	22.41%	
Second World War	6	17.50%	0.00%		
Grand Total		**55.16%**	**43.87%**	**7.48%**	**6.62%**

Table 6: Average precisions for historical context queries by historical period, context type, and model.

Blank cells indicate values that could not be computed due to less than ten poems existing for that period, except in the case of the Great Depression, where Gemini produced malformed responses that caused a divide-by-zero error when computing the average precision.

Historical *authorship* context queries (those which ask for poems by authors of a certain period) showed the best average precision across all query types on both models: 55.16% for

Gemini and 43.87% for Llama. Conversely, historical *publication* context queries (those which ask for poems published during certain periods) showed the worst: just 7.48% for Gemini and 6.62% for Llama. This seems paradoxical — the task of seeing if two ranges (the author lifespan and the historical period) overlap is more logically complex than checking if a single year (the publication year) lies within a range (the historical period). But I suspect that this again reflects a pattern in the training data of each model. Poets often have biographies available online which situate them in a certain historical period; for example, William Wordsworth's biography on the Poetry Foundation website describes him as "one of the founders of English Romanticism," which immediately places him within the Romantic era that occurred in Europe in the early 1800s ("William Wordsworth"). Individual poems typically do not have these types of informational texts unless they were particularly influential, so historical associations for lesser-known poems are largely absent from the training data. This would make the task of finding relevant authors easier than finding relevant poems, which is the pattern I observed.

But training data was likely not the only factor at play either. There existed a moderately strong correlation ($r = 0.6$ for Gemini, 0.58 for Llama) between the length of the historical period in years and the average precision on authorship context queries, meaning that the longer the period, the more precise the models were in their results. This would indicate that chance plays a role as well; selecting poems at complete random, you're more likely to choose poems that share a 300-year range of publication than a 4-year one.

Dataset Coverage

A performance issue that plagued the entire benchmark on both models was the fact that the models' responses would cluster strongly around a small subset of poems in the dataset. This is perhaps the most severe flaw in model behavior for our use case, as classical search methods

would weigh each poem much more evenly. In a dataset of 1,453 poems, Gemini only returned 114 unique poems, and Llama only returned 122. These subsets only cover 8 - 8.5% of the dataset as a whole.

Poem Title	Author	Number of Tags	Occurrences
Elegy V: His Picture	John Donne	13	232
Elegy to the Memory of an Unfortunate Lady	Alexander Pope	6	174
from The Prelude: Book 1: Childhood and School-time	William Wordsworth	9	148
The Hosts	Alan Seeger	3	130
To a Reason	Arthur Rimbaud	3	87

Table 7: The top five most frequently returned poems for Gemini, with number of tags.

Poem Title	Author	Number of Tags	Occurrences
from The Prelude: Book 1: Childhood and School-time	William Wordsworth	9	317
Elegy V: His Picture	John Donne	13	278
Elegy to the Memory of an Unfortunate Lady	Alexander Pope	6	261
The Hosts	Alan Seeger	3	153
The Rape of the Lock: Canto 5	Alexander Pope	8	130

Table 8: The top five most frequently returned poems for Llama, with number of tags.

The most commonly returned poem by Gemini was “Elegy V: His Picture” by John Donne, owed partly to the fact that it was duplicated in the dataset (as it is hosted by both the Poetry Foundation and the Academy of American Poets). The most frequent poem in Llama’s responses (and the third most frequent in Gemini’s) was the first book of William Wordsworth’s “The Prelude,” called “Childhood and School-time.” Other exceedingly common poems returned included “Elegy to the Memory of an Unfortunate Lady” by Alexander Pope, “The Hosts” by Alan Seeger, and “To a Reason” by Arthur Rimbaud.

It’s not immediately clear why these poems were returned so frequently by both models. The correlation between number of tags and occurrences was very small ($r = 0.06$ for Gemini, 0.11 for Llama), so it’s not that these poems span many forms, techniques, themes, or historical

periods. It likely comes down to emergent patterns in the models' training datasets that mean some poems are weighed much more heavily than others. These poems may be more strongly associated with the concept of a "poem," with the text in the system prompt, or they may simply appear more frequently in the training data — it's difficult to know. More research is needed surrounding the qualities of these poems and their prevalence in literature and culture.

Malformed Responses

Another issue that impacted performance on the benchmarks was the generation of malformed responses by the models in response to certain queries. A malformed response is defined as a response that is not strictly a comma-separated list of seven-character alphanumeric strings. For example, "P778356, P778387, P778340, P778355" would be considered a well-formed response, but "Here are your poems: P0001, P123456, 2000, poem_1" would not. 9.6% of Gemini's responses were malformed, whereas only 5.4% of Llama's were.

	Gemini		Llama	
Query Type	Well-formed	Malformed	Well-formed	Malformed
Form	31	25	49	7
Technique	18	2	19	1
Theme	375	17	374	18
Historical Author Context	25	5	29	1
Historical Pub Context	12		11	1
Grand Total	**461**	**49**	**482**	**28**

Table 9: Malformed response counts for each model by query type.

Malformed responses took four major forms: refusal, total hallucination, verbosity, and collapse. Responses exhibiting refusal show the model declining to carry out the task for some reason. An example is "I am sorry, but I cannot fulfill that request. None of the documents provided appear to be haiku poems." In instances of total hallucination, the model generates a list of poem IDs that bear no resemblance to those in the dataset. For example: "1360, 0811, 0153,

0342, 0432, 1341, 0272, 0857, 1159, 0858” is a response Gemini gave to “examples of concrete/pattern form.” A model exhibits verbosity when it adds unnecessary text to its response that prevents us from parsing it easily, usually in the form of a preamble. An example of this is “Based on the provided sources, the following poems match the search query 'american civil rights movement authors': P777570, P778245.” We can see that this contains a valid list of poem IDs, but the model did not adhere to its instructions of only producing the list of IDs by itself. Finally, model collapse is when the model gets trapped in a loop of generating text until some timeout is reached, which took up to 82 seconds for some queries in the benchmark.

Only Llama experienced collapse during the benchmark, and did so several times. Likewise, only Gemini exhibited refusal, and its repeated refusal for some queries affected its performance metrics significantly. Both models produced instances of total hallucination and verbosity during their benchmarks.

Discussion

Previous Research

This research aligns with several hypotheses and findings from the Walsh et al. study. Most significantly, we reinforce the finding that LLMs are more capable of classifying "fixed" poetic forms than unfixed forms or "formal elements" (defined in this paper as *techniques*; Walsh et al. 6). As in the Walsh study, our models were the most capable at identifying sonnets, but only in this study did they perform poorly at identifying haiku. Our models' lack of precision on haiku queries is likely due in part to the fact that haiku do not typically contain the word "haiku" in their titles, and that many of the haiku in the dataset were translated into English and thus did not retain the 5-7-5 syllabic composition that is typical of the form. Still, 0% precision on all haiku queries (and on many other form queries) was a remarkable result that indicated something about this particular context which prevented the models from identifying any attributes of the form at all. Similarly, we found that elegy was the most precisely identifiable form among the unfixed forms, as it was in the Walsh study, but the models struggled to identify ekphrasis and completely failed to identify ars poetica in the dataset. Our results did, however, agree on prose poems, odes, and concrete/pattern poems being particularly difficult to identify; our models were not able to retrieve any examples of the latter two, and performed very poorly (5-10% precision) on retrieving prose poems.

Hypotheses on Performance

There may be a number of reasons why performance on this task was so poor overall. The most likely is a limitation of current retrieval-augmented generation methods. RAG relies on an attached vector database of documents from which individual documents are retrieved based

on vector similarity (Choi et al.). This works well in instances where the document text is semantically similar to the prompt (for example, asking "what are rocks made of?" and getting a document titled "The Composition of Rocks"). In our case, natural-language search queries are usually quite dissimilar from the poems' content, which means the distance between their vectors will be larger (and thus they will be considered less similar). We're relying on the vector embedding of the poems to contain meaningful formal and technical information that is connected to words in the query; as one example, we're hoping that "haiku" is judged as statistically similar to the presence of three lines comprised of 5, 7, and 5 syllables. But whether or not this is actually encoded depends entirely on the makeup of the training data and the particular algorithms used to train and fine-tune the model.

Additionally, the *tokenizer*, which is the component of a LLM that converts text into vector embeddings, doesn't always encode formal information in predictable or intuitive ways. Using OpenAI's interactive tokenizer, we can observe patterns in how text is converted to a vector representation for a major LLM ("Tokenizer"). We'll use a haiku by Matsuo Bashō as an example:

> 'Tis the first snow—
> Just enough to bend
> The gladiolus leaves!

This poem gets split into the following tokens (bounded by square brackets):

> ['T][is][the][first][snow][—]
> [Just][enough][to][bend]
> [The][gladi][olus][leaves][!]

As we can see, each single space is absorbed into the next token instead of standing alone. The function of the space is not considered significant enough to constitute its own unit of meaning. Additionally, new lines are not encoded at all; only double line breaks get their own

token (double spaces are also encoded separately). This presents a problem for poetry embedding, as visual blocking and whitespace play a major role in identifying forms and techniques. I predict that this is the reason Walsh et al. found that models struggle to identify forms which are distinguished by their use of whitespace, like concrete/pattern poems and prose poems (Walsh et al. 7). It could also explain why we saw such poor performance on techniques like alliteration — the individual letters and sounds do not have their own representations in the vector form. Thus, necessary information is lost during this process which can affect how the model perceives the form of a poem.

Feasibility as a Search Engine

In its current state, RAG is suitable for implementing capable search functions for many types of unstructured data. Poetry does not appear to be one of them. Benchmark results showed low precision and especially low recall across most queries, which users and database maintainers alike would consider unacceptable. Additionally, for such poor performance, LLMs use up extraordinary amounts of compute compared to more classical search algorithms, so there is no cost-benefit analysis which would choose LLMs over existing solutions to the document retrieval problem.

Still, there were various indicators of promise throughout the benchmark data. The average MRR indicated that the first correct result was very often the first or second overall result, which is a quality we'd expect from any search algorithm. Additionally, the latency of the models' responses was not unreasonable at around 1.3 seconds on average. This time gap could be bridged by some kind of animated spinner or loading UI that keeps the users' attention. Second-plus load times are a commonly encountered (and thus widely mitigated) user experience across all types of applications.

Limitations

As the present research is based entirely off the dataset released by Walsh et al., I share most of the limitations that they set out in their paper. All poems in the dataset are English-language poems published in North America and Europe, as these are the poems primarily represented by the collections of the Poetry Foundation and Academy of American Poets. This precluded me from testing a wider, more representative sample of poetry as a literary genre. Walsh et al. point out that these collections are not "comprehensive or representative (in terms of gender, race, culture, geography)," and possess "an uneven distribution of poems in each form, reflecting biases related to race, class, language, and culture." These biases are likely reproduced by the LLMs, which pose "a risk of reinforcing dominant understandings of poetic form and prosody" (Walsh et al. 10).

Additionally, as noted before, human custodians of poetry databases are not exceptionally accurate at classifying poems themselves. I have no insight as to how the tags on these poems were applied, and often found during testing that the models returned poems that *were* relevant to a query, but had not been tagged appropriately, and thus were marked as incorrect. For example, when searching "poems written about love," Gemini returns Petrarch's 102nd sonnet, which starts:

> If no love is, O God, what fele I so?
> And if love is, what thing and which is he?
> If love be good, from whennes cometh my woo?

Since this poem is tagged as "Heartache" instead of "Love," it is marked as incorrect for that query, but would likely be considered acceptable to a real person. Significantly worse is that some poems are not tagged with their correct *forms*, even when these forms are obvious; Walsh

et al. found a series of haiku hosted by the Poetry Foundation which were not tagged as such (Walsh et al. 10).

Another significant limitation I faced was the size of the dataset. Walsh et al. originally compiled a larger dataset containing a significant portion of in-copyright works, then distilled it down to only public domain poems when releasing their data. This resulted in many of the forms, techniques, themes, and historical periods I wanted to test having fewer than ten relevant poems present in the set. After generating the queries, I was only able to represent 14 forms (3 fixed, 11 unfixed), 10 techniques (5 fixed, 5 unfixed), 98 themes, and 10 historical periods in the data. Additionally, I was constrained by the number of ways to phrase each query type, so each form only generated 4 queries, each technique generated 2, each theme generated 4, and each historical period generated 6 (3 for authorship, 3 for publication). This meant that a lot of the averages computed to summarize the performance of the models were highly sensitive to performance on individual queries and would likely fluctuate between runs of the benchmark. They are almost certainly not representative of the models' theoretical performance on a larger dataset containing in-copyright poems.

The benchmark was also computationally intensive, and I was subject to strict usage quotas throughout the project. Google Cloud fixed a quota of five queries per minute on my project, which meant that each benchmark took at least an hour and 42 minutes, plus the sum of all the latencies in that test run. This prevented me from being able to experiment with different system prompts and configurations efficiently. It is likely that the exact configuration of the benchmark could be optimized to achieve better results overall, with significant improvements in certain areas, but it would still be subject to the technical limitations of RAG and the idiosyncrasies of each model.

Conclusion

At the current juncture, it doesn't appear that large language models could feasibly be used in poetry databases to replace classical keyword-based search algorithms any time soon. They fail by most metrics used to evaluate information retrieval systems, exhibiting poor precision, recall, and F1 scores. Additionally, they cluster around a small subset of poems according to their training data, which makes them unhelpful for large datasets which require wide coverage. These failures, plus the models' computational inefficiencies, make LLMs an obvious 'no' for any application involving poetry retrieval.

Still, there is evidence to believe that the technology is headed in the right direction for this use case; the models are sufficiently speedy and capable of ranking results in ways that would be useful in a search context. As large language models get faster, more efficient, and more capable of advanced reasoning, it's possible that they will be able to recognize formal, technical, thematic, and historical attributes of poems much better. Future research should examine the exact ways in which these attributes may or may not be expressed in embedded vectors, and expand the size and diversity of the poems and poetic attributes being tested. Furthermore, a fine-tuned or purpose-built language model for poetry classification and retrieval could demonstrate dramatically improved results on this benchmark. Despite what LLM marketers would like you to think, poetry continues to pose a challenging problem in the field of natural language processing. It is nonetheless a fascinating one, and a new angle from which to appreciate the art form.

Bibliography

Bashō, Matsuo. "['Tis the first snow—]." Poets.org, Academy of American Poets, 1899, https://poets.org/poem/tis-first-snow. Accessed 11 May 2025.

Binder, Jeffery M. "The Datafication of Culture: Romanticism and AI-Generated Poetry." The Wordsworth Circle, vol. 53, no. 3, 2022, pp. 354-373. The University of Chicago Press Journals, https://doi.org/10.1086/720908.

Boonstra, Lee, and Google. "Prompt Engineering." 2025. Kaggle, https://www.kaggle.com/whitepaper-prompt-engineering.

Cao, Bowen, et al. "On the Worst Prompt Performance of Large Language Models." 2024. arXiv, https://arxiv.org/abs/2406.10248v1.

Choi, Nicole, et al. "What is retrieval-augmented generation, and what does it do for generative AI?" The GitHub Blog, 4 April 2024, https://github.blog/ai-and-ml/generative-ai/what-is-retrieval-augmented-generation-and-what-does-it-do-for-generative-ai/. Accessed 11 May 2025.

Ge, Jin, et al. "Development of a Liver Disease-Specific Large Language Model Chat Interface using Retrieval Augmented Generation." Hepatology, vol. 80, no. 5, 2024, pp. 1158-1168. https://doi.org/10.1097/hep.0000000000000834.

Google. "Use the Gemini Web App." Google Help, https://support.google.com/gemini/answer/13275745. Accessed 24 November 2024.

Google. "Write with Gemini in Google Docs." Google Help, https://support.google.com/docs/answer/13951448. Accessed 24 November 2024.

Peng, Ruoling, et al. "Embedding-Based Retrieval with LLM for Effective Agriculture Information Extracting from Unstructured Data." 2023. arXiv, https://arxiv.org/abs/2308.03107.

Petrarch. "Sonnet 102 [If no love is, O God, what fele I so?]." Poets.org, Academy of American Poets, 1375, https://poets.org/poem/sonnet-102-if-no-love-o-god-what-fele-i-so. Accessed 10 May 2025.

"Tokenizer." OpenAI Platform, OpenAI, https://platform.openai.com/tokenizer. Accessed 11 May 2025.

Walsh, Melanie, et al. "Sonnet or Not, Bot? Poetry Evaluation for Large Models and Datasets." 2024. arXiv, https://arxiv.org/abs/2406.18906.

"William Wordsworth." Poetry Foundation, https://www.poetryfoundation.org/poets/william-wordsworth. Accessed 10 May 2025.

Wu, Shirley, et al. “STARK: Benchmarking LLM Retrieval on Textual and Relational Knowledge Bases.” 2024. arXiv, https://arxiv.org/abs/2404.13207.

Zhang, Ran, and Steffen Eger. “LLM-Based Multi-Agent Poetry Generation in Non-Cooperative Environments.” 2024. arXiv, https://arxiv.org/abs/2409.03659.