The Pneuma Project: Reifying Information Needs as Relational Schemas to Automate Discovery, Guide Preparation, and Align Data with Intent

Muhammad Imam Luthfi Balaka
luthfibalaka@uchicago.edu
The University of Chicago
Chicago, USA

Raul Castro Fernandez
raulcf@uchicago.edu
The University of Chicago
Chicago, USA

ABSTRACT

Data discovery and preparation remain persistent bottlenecks in the data management lifecycle, especially when user intent is vague, evolving, or difficult to operationalize. The Pneuma Project introduces Pneuma-Seeker, a system that helps users articulate and fulfill information needs through iterative interaction with a language model–powered platform. The system reifies the user's evolving information need as a relational data model and incrementally converges toward a usable document aligned with that intent. To achieve this, the system combines three architectural ideas: context specialization to reduce LLM burden across subtasks, a conductor-style planner to assemble dynamic execution plans, and a convergence mechanism based on shared state. The system integrates recent advances in retrieval-augmented generation (RAG), agentic frameworks, and structured data preparation to support semi-automatic, language-guided workflows. We evaluate the system through LLM-based user simulations and show that it helps surface latent intent, guide discovery, and produce fit-for-purpose documents. It also acts as an emergent documentation layer, capturing institutional knowledge and supporting organizational memory.

CCS CONCEPTS

• **Information systems** → **Data management systems**; **Information retrieval**; • **Human-centered computing**;

KEYWORDS

Data Discovery, Data Preparation, Information Needs, Large Language Models, Human-Computer Interaction

1 INTRODUCTION

Large Language Models (LLMs) are poised to support the entire data management lifecycle, from collection to task execution [8, 40, 38, 18]. Among the stages that have most resisted automation are data discovery, the task of identifying and retrieving documents relevant to a user's *information need* [4, 7], and data preparation, the transformation of those documents into a usable form for downstream tasks. A key challenge lies in the nature of the user's information need: it is often vague, evolving, and difficult to express in a way that software can act on effectively. As LLMs become increasingly capable, the bottleneck shifts from executing tasks to helping users articulate their goals—their information needs—precisely enough to be fulfilled with available data.

Consider a practical example from our university's Finance department: *"What impact will tariffs have on our organization?"* Answering this question requires addressing several steps:

(1) Discover relevant data sources, such as current tariff schedules and procurement records;
(2) Define "impact," which may include both direct (e.g., imported goods from tariffed countries) and indirect effects (e.g., tariffed components in otherwise unaffected imports);
(3) Determine temporal scope: Is the user interested in projected impact next week, next fiscal year, or in a retrospective analysis?;
(4) Integrate these data sources into a coherent document that supports meaningful interpretation.

To support such inquiries, a system must extract, make explicit, and operationalize the assumptions behind the question. This means surfacing the "information need," which is the set of *states of nature* required to answer the user's question [3]. This articulated information need then guides discovery, integration, and preparation. In practice, these processes are rarely executed by a single user; modern organizations involve business analysts, data scientists, and data engineers collaborating in a semi-decentralized way to answer complex questions [36].

Pneuma-Seeker is designed to help users articulate and fulfill such information needs semi-automatically. A central insight behind Pneuma-Seeker is that an information need can be reified as a data model, specifically, a relational schema. The system then seeks to align that schema with available data, returning a document that satisfies the latent information need. Initially, neither the system nor the user may know exactly what this document looks like, but through interaction, both sides refine their understanding, e.g., the user may not ask to include the effect of direct and indirect tariffs initially, but expresses this explicitly after seeing an intermediate output. The user offers language-based feedback, and the system proposes increasingly aligned data models, converging over iterations toward one that meets the desired information need.

When the user asks Pneuma-Seeker the tariff question, the system examines a procurement database with dozens of tables, determines that tariff information is missing, retrieves relevant data from online sources, integrates the information into a tabular structure, and proposes a preliminary document for review. After two rounds of user feedback, the system converges on a schema and SQL query that estimates tariff exposure from German suppliers, matching the user's evolving understanding of "impact."

 16th Annual Conference on Innovative Data Systems Research (CIDR '26). January 18-21, Chaminade, USA

LLMs in Pneuma-Seeker serve as a bridge between language, which users employ to express intent, and data models, which the system uses to steer discovery and preparation. Pneuma-Seeker leverages recent advances in retrieval-augmented generation (RAG) [16] and agentic architectures to make this bridge actionable. To enable this functionality, Pneuma-Seeker introduces three generalizable contributions:

- **Context Specialization.** LLMs struggle with large, diverse contexts in structured data tasks [15, 19, 17]. Pneuma-Seeker addresses this by decomposing the workflow into subtasks with specialized contexts and narrow scopes.
- **Conductor-Style Planning.** Rather than static, rule-based pipelines, Pneuma-Seeker employs a conductor component, an LLM-powered agent that assembles plans dynamically, based on real-time evidence of progress toward fulfilling the information need.
- **Convergence Criteria.** Pneuma-Seeker treats the evolving data model as a shared state between user and system: the user refines it via language, and the conductor uses it to guide downstream modules. Convergence occurs when the document aligns with the user's latent information need.

To evaluate Pneuma-Seeker without human subjects, we use LLM Sim, an LLM-based agent that acts as users, reacting only to system outputs. While not a full substitute for real studies, both quantitative and qualitative results show Pneuma-Seeker helps articulate and fulfill information needs.

Pneuma-Seeker's design offers an important side effect: it acts as a mechanism for organizational knowledge capture. By prompting users to articulate their goals, it surfaces tacit information needs and latent dataflows. At scale, these interactions accumulate into emergent documentation, preserving tribal knowledge and institutional memory. As teams evolve, Pneuma-Seeker strengthens the resilience of internal data ecosystems by making this knowledge explicit and reusable.

2 FROM INFORMATION NEEDS TO ANSWERS

In this section, we define relevant concepts, survey the landscape of solutions that help users satisfy those needs, and present an interaction model that motivates the design of Pneuma-Seeker. We aim to highlight recurring challenges in how users articulate and refine their information needs, and to motivate a structured approach in which human information needs and system representations co-evolve toward the latent information need.

2.1 Definitions

Information Need. An information need is the set of states of nature required to solve a data-driven task [3]. This definition is general: an information need may include the features required for training a classifier, the schema to answer a SQL query, or the variables for causal inference. These states are often encoded in documents. For clarity, we assume one information need per task, though multiple valid representations may exist.

Latent and Active Information Needs. The latent information need is the true set of states needed to solve a task, often initially unknown to the user. The active information need is the user's working hypothesis about what data is needed, which evolves through interaction and exploration to approximate the latent one.

2.2 Landscape of Solutions

Document Retrieval. Once an active information need is specified, a wide array of technologies can help retrieve relevant documents. For example, web search engines map keyword queries to matching documents. In structured databases, SQL queries retrieve relations containing specific columns, assuming join paths exist. RAG systems enable LLMs to retrieve relevant text chunks from vector stores or databases before composing answers. These retrieval methods assume that the information need is reasonably well articulated, which is often the core bottleneck [2, 13, 33].

Identifying Information Needs. A separate but critical challenge is helping users formulate their information needs in the first place. Many user interfaces offer scaffolding to support this process: web autocomplete features leverage population-level priors; LLM interfaces often suggest next actions or clarifying prompts; enterprise data catalogs [12] surface usage patterns (e.g., users querying table A and B often also use table C), aiding newcomers in forming expectations about what data is relevant.

Representing Information Needs. The notion of an information need has long been studied, especially in information retrieval (IR) [2, 28]. In IR, an information need is often framed as the user's underlying goal, which a query only imperfectly expresses. In exploratory search, users engage in open-ended, investigative information seeking where goals evolve during the search process [22, 34]. Similarly, in exploratory data analysis (EDA), the focus shifts to iterative refinement and hypothesis evolution, often guided by visualization [30, 11]. In the Pneuma project we build on these lines of work to provide computational representations of information needs that human users and machines can co-evolve.

In this work, we reify an information need as a relational data model plus a SQL query over that model. This choice reflects our view that solving a data-driven task ultimately involves instantiating a structured document (a table or set of tables) that aligns with the user's intent. The system's job, then, is to identify the schema and query that materialize the latent information need.

2.3 Aligning Data with Intent: A Model

We propose the following model to guide system design. A human user seeks to solve a data-driven task. That task induces a latent information need, which, if represented as a document, would suffice to solve the task. However, the user may not initially know what that document looks like.

An abstract system, initially agnostic to the task, holds an internal state representing its evolving understanding of the information need. The interaction proceeds in iterations: the user communicates their active information need, and the system updates its state accordingly. After observing the system's output, the user revises their input, gradually steering the system closer to the latent information need. The interaction concludes when the system produces a document (or schema + query) that the user deems sufficient.

In contemporary practice, this system is not software alone—it is a composite of human roles and infrastructure: business analysts, data scientists, data engineers, domain experts, databases, interfaces, APIs, and more [36]. With Pneuma-Seeker, we aim to unify these roles under a coherent software system that helps users identify, articulate, and materialize their information needs through iterative, language-guided interaction.

3 THE PNEUMA-SEEKER SYSTEM

In this section, we describe Pneuma-Seeker, a system that helps users identify and fulfill their latent information needs. We first present its architecture and key design principles, followed by a detailed explanation of each component.

3.1 Technical Contributions and Overview

Pneuma-Seeker is designed around three technical insights:

Context specialization. When working with structured data, an LLM is constrained by context size heterogeneity. The more heterogeneous the context, the more attention is spread across unrelated details. In addition, prompting an LLM with distinct *roles* can help focus its behavior [29]. Thus, Pneuma-Seeker adopts a multi-component architecture in which retrieval, integration, and orchestration are separated. Each component focuses on a single subtask, and the LLM receives only the information relevant to its assigned role.

Dynamic planning with Conductor. The subtasks induced by context specialization must still be assembled into an end-to-end process of identifying and fulfilling a user's information need. Rather than relying on a static, rule-based pipelines, Pneuma-Seeker uses Conductor, an LLM-powered agent that orchestrates the process based on real-time evidence of progress toward fulfilling the information need.

Shared state for convergence. Pneuma-Seeker represents a user's active information need as a relational data model (T, Q), where T is a set of tables, and Q is a sequence of SQL queries over T. The user and Pneuma-Seeker establish a feedback loop: the user describes their active information need as it changes to the system, the system updates (T, Q), and the user reacts to those changes with additional feedback. The interaction ends when the user stops or when the active information need matches the latent information need (i.e., convergence).

These three insights directly shape Pneuma-Seeker's modular architecture. Figure 1 shows the architecture, which consists of multiple components: **(1) Conductor**, which orchestrates the process; **(2) IR System**, which retrieves relevant data; and **(3) Materializer**, which integrates and prepares data to form T. We describe them in detail below.

3.2 Conductor

Conductor drives Pneuma-Seeker towards convergence by selecting actions on the fly to align (T, Q) with a user's active information need. At the moment, it selects actions one at a time, but there has been some research on parallelized LLM planning (e.g., [37, 41]). In Conductor, an action is any of the following:

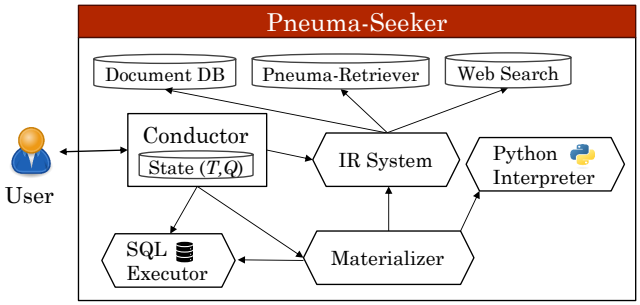

Figure 1: The Architecture of Pneuma-Seeker

- **Internal reasoning.** Conductor engages in internal reasoning (inspired by ReAct [35]), where we prompt the LLM so that it is able to evaluate the current state (T, Q), retrieved data from IR System, and a user's most recent feedback, to decide the best next action(s). For example, suppose the user clarifies that determining the effect of a new tariff requires knowing the previous active tariff. Conductor might reason: *"My current query computes price change using only the new tariff percentage. I should retrieve the previous tariff percentage and update the final percentage to (new_tariff – previous_tariff)."*
- **Tool call.** Conductor can directly call tools, including IR System, Materializer, and SQL Executor, or invoke them after first identifying the need through internal reasoning. For example, if it determines that the previous active tariff is required, it may issue a retrieval request to IR System: *"Retrieve the previously active tariff for the region."* Similarly, if it decides that T should be materialized, it may invoke Materializer with a note such as: *"Materialize T, a table containing relevant procurement data with new and previously active tariff information for the supplier country."*
- **State modification.** Conductor can update the state (T, Q) to revise interpretations of the user's active information need. It may modify only T, only Q, or both. For example, after determining that Q should account for the previous active tariff, it may update Q to look like this: *["SELECT price * (1 + new_tariff - previous_tariff) AS new_price FROM procurement_data"]*.
- **User-facing communication.** Conductor can interact directly with the user to explain actions taken, summarize the current state, ask clarifying questions, or propose next steps. For instance, after executing the queries in Q, it may inform the user: *"I have executed the queries in Q. The new price for items bought from supplier 12345 in Germany is 5% higher, with an average increase of $5,125."*

By design, Conductor operates without fixed procedural rules, aside from essential dependencies (e.g., T must be materialized before executing Q). Conductor limits the number of consecutive actions to a fixed value i, which we set to $i = 5$ based on empirical observation that the LLM rarely hits this limit during our evaluation. This limit is intended to prevent (T, Q) from moving away from the latent information need before user feedback can correct it, while also avoiding long autonomous runs that keep users waiting. In addition, we instruct Conductor to end each sequence of actions

with a user-facing message whenever possible. If the action limit is reached without producing a user-facing message, the system interrupts and forces Conductor to do so.

When interacting with the user, Conductor grounds its decisions on data retrieved from IR System, rather than relying solely on assumptions (e.g., assuming we have suppliers with ID 12345 in a procurement table). This avoids producing unrealistic (T, Q) states that cannot be materialized. For instance, if the user requests an analysis of the impact of a new tariff for a specific supplier in 2019, but IR System only retrieves tariff records from 2020 onward, Conductor can detect the gap and either search alternative sources or notify the user, rather than spending multiple actions building and materializing a (T, Q). This prevents wasted effort and keeps the interaction focused on attainable outcomes.

Throughout an interaction, Pneuma-Seeker surfaces not only user-facing responses but also the current state (T, Q) to the user. We display the interface in Figure 2 (as of August 2025). Displaying the state (box 3; sample rows are shown for T) alongside the chat interface (box 1 and box 2) allows users to spot subtle mismatches between their intent and the system's evolving model and correct them. Continuing the tariff-impact example, suppose T contains procurement data but lacks the `country` attribute entirely, even though the user only cares about German suppliers. Without this attribute, Q cannot filter the data correctly, and the final computation would be misaligned with the user's intent.

3.3 Information Retrieval (IR) System

IR System supports Conductor and Materializer by retrieving relevant data from multiple sources. It abstracts heterogeneous retrieval format, such as tables and text, into document objects. This uniform representation allows new retrievers to be added without changing the rest of IR System's design. Nevertheless, both Conductor and Materializer knows about the kind of data available in the system: currently tables, domain knowledge, and web pages, which are handled by the following three retrievers in our current implementation:

- **Pneuma-Retriever [1]**, a state-of-the-art table discovery system with a hybrid index, combining an HNSW [21]-based vector store and a BM25 [27]-based inverted index for efficient table search.
- **Document Database**, which uses Pneuma-Retriever's indexer to store domain knowledge. This enables cross-user knowledge transfer. For example, if one user specifies that estimating tariff impacts requires accounting for both direct and indirect tariffs, subsequent tariff-related queries can leverage that insight. Pneuma-Seeker automatically captures knowledge from user interactions and save it to Document Database, inspired by prior work (e.g., [24, 31, 20]).
- **Web Search**, which provides a thin interface to external search engines for general or up-to-date information lookup.

With the current implementation, Pneuma-Seeker can already handle complex users' information needs that combine structured and unstructured sources. Building on this design, we will extend IR System to accept direct feedback on retrieved documents from Conductor or Materializer.

3.4 Materializer

Materializer's sole purpose is to populate T with data, possibly involving integration of multi-source data from IR System. Materializer is separate from Conductor, reflecting the **context specialization** principle: Materializer operates only on context relevant to data integration and transformation, without being distracted by orchestration details. Materializer also considers Q, so it understands filters in the queries. For example, if a query in Q expects a `date` column to be of format "yyyy-mm-dd," while the column actually lists its values with format "Month Day, Year," Materializer will transform the values of this column by producing Python code.

Materializer's toolkit includes a DuckDB [26]-based SQL Executor for standard relational operations (joins, unions, etc.) and a Python interpreter equipped with Pandas [23] and NumPy [10]. Similar to IR System, Materializer is designed to be extensible, allowing new operators to be incorporated easily, including semantic relational operations (e.g., as defined by LOTUS [25]).

Through its prompts, Materializer is aware of all retrieved documents from IR System and can leverage them to form SQL queries or Python code. For example, if Materializer retrieves two documents representing (1) the latest tariffs from Web Search and (2) internal procurement tables from Pneuma-Retriever, then it can generate Python code that combines both documents (e.g., tariff information becomes a new column in the table). Errors may occur (e.g., floating-point operations on columns with null values), so the respective tool analyzes these errors and provides feedback to Materializer to fix the generated queries or code.

We plan to explore interaction models in which Conductor provides incremental feedback to Materializer after every n operations, mirroring the way users interact with Conductor. This additional supervision could help prevent Materializer from drifting away from the intended T, especially in long integration pipelines.

3.5 Dynamic vs. Static Pipelines

In this section, we reflect on previous designs we experimented with and justify Pneuma-Seeker's design based on that experience.

We design both Conductor and Materializer is significantly more adaptable for incorporating more actions/tools (and hence cover more use cases) compared to the static pipeline we initially experimented with—define (T, Q), retrieve top-k tables, filter and integrate the tables via relational operations, and prune the integration results down to those defined in T. While such a static pipeline can cover a variety of use cases, it becomes harder to update as we incorporate more actions/tools. Whenever a step in the original pipeline is insufficient (e.g., lacking the actions needed to cover a new use case), the entire pipeline must be revisited.

For example, in our tariff-impact scenario, there may be cases where the correct analysis requires joining procurement data with tariff records, but the sources do not share a common identifier (e.g., `supplier_id`). In such cases, the system needs to employ more flexible relational operations (e.g., semantic or fuzzy joins) that were not part of the original pipeline. Incorporating such an operation into a static, predefined pipeline may require significant engineering effort, since its position in the pipeline must be chosen

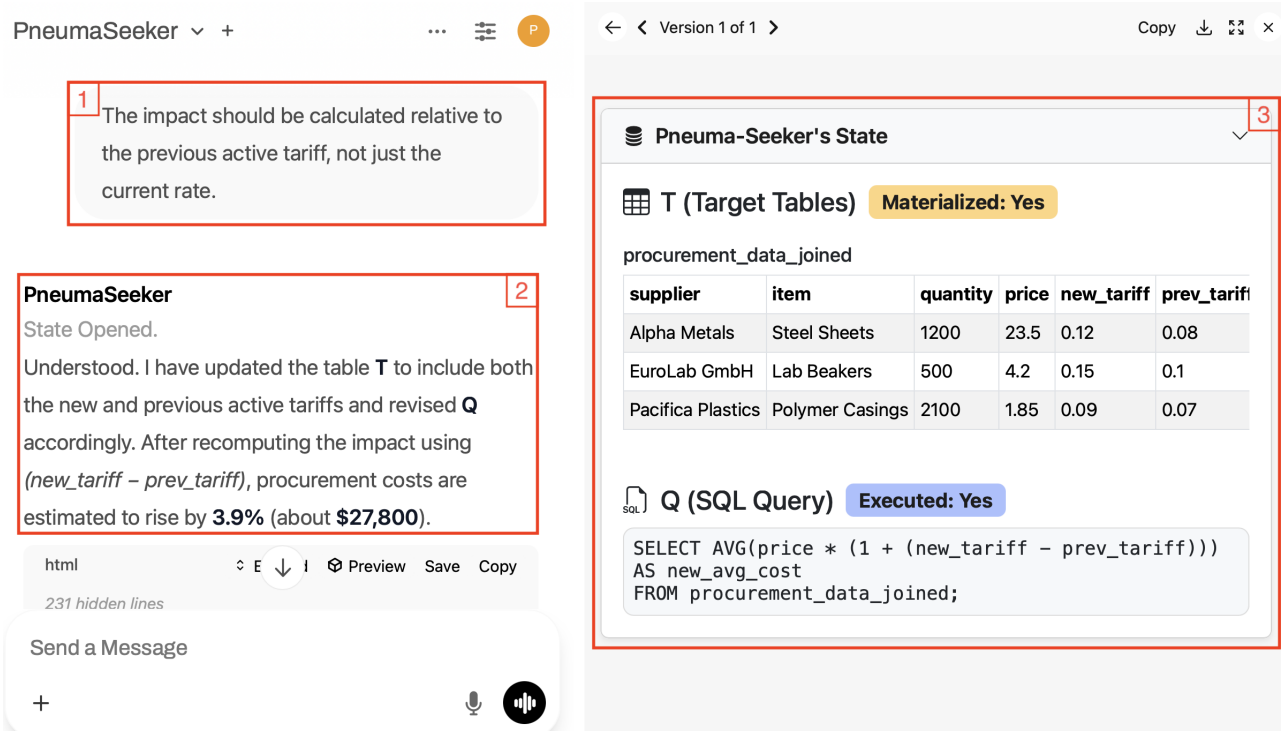

Figure 2: Interface of Pneuma-Seeker, showing: [1] User Query (Clarification), [2] User-Facing Message, and [3] State View Page (T, Q). Note: the numbers and values of T shown here are not real for privacy reasons.

carefully and its assumptions must align with both upstream and downstream operations.

In other words, generalizing a static pipeline to support a wide variety of use cases is possible, but every extension increasingly requires substantial effort. As more actions/tools are added, these integration costs compound. In contrast, with Conductor or Materializer, we simply define the new operation or tool, and it naturally fits into their action spaces.

3.6 Pneuma-Seeker in Action

To illustrate how the components work together, we revisit the finance department's broad question from Section 1: *"What impact will tariffs have on our organization?"*
During the interaction, the user adds a key clarification: *"Impact should be calculated relative to the previous active tariff, not just the current rate."*

Conductor first engages in **internal reasoning**, recognizing that it needs both the new and previous active tariff information to compute the impact correctly. It then issues a tool call to **IR System**, which returns procurement tables from Pneuma-Retriever and tariff information from the Web Search interface. Based on this, Conductor defines T as a single table combining procurement data with tariff information. It summarizes the retrieved sources and asks the user: *"I found procurement records and historical tariff information for the relevant suppliers. I've designed T to combine the tariff information with the procurement data. Should I materialize T now?"*

Once the user confirms, Conductor calls **Materializer** to join the procurement and tariff data, thereby populating T. With T materialized, Conductor performs a **state modification** to update Q so that it computes cost impact relative to the previous tariffs:

```
["SELECT AVG(price * (1 + (new_tariff - prev_tariff)))
 AS new_avg_cost FROM procurement_data_joined"]
```

SQL Executor runs the updated query and returns an estimated x% increase in procurement costs (about $y) across all goods.[1] Conductor reports this information to the user through **user-facing communication**:

"If the new tariffs take effect, procurement costs are estimated to rise by x% (about $y), calculated relative to the previous rates."

At this point, the user may further narrow the scope, e.g., to only consider lab equipment and suppliers from Germany, trigger another refinement cycle.

4 EVALUATION

In this section, we offer preliminary evidence of Pneuma-Seeker's ability to help users articulate information needs and solve data tasks by answering the following questions:

- **RQ1**: Can the user reach their underlying information need by interacting with Pneuma-Seeker?
- **RQ2**: Given a specific information need, can Pneuma-Seeker address it accurately?

[1]The actual numbers are not disclosed for privacy reasons.

Evaluation Workload. We evaluate Pneuma-Seeker on the archaeology and environment datasets from KramaBench [14], with Web Search disabled to prevent leaking benchmark information from the internet. These datasets are associated with 12 and 20 questions, respectively. Table 1 shows the datasets' characteristics.

Table 1: Characteristics of the Datasets

Dataset	# Tables	Avg. #Rows	Avg. #Cols
Archeology	5	11,289	16
Environment	36	9,199	10

Each benchmark question is a latent information need. We use an LLM (`GPT-4o`) to simulate a domain expert (LLM Sim) interacting with the system. Starting from a broad prompt, LLM Sim iteratively refines its active information need based on the system's outputs. For example, given the latent question:

"What is the average Potassium in ppm from the first and last time the study recorded people in the Maltese area? Assume that Potassium is linearly interpolated between samples. Round your answer to 4 decimal places."

The initial query is:

"I'm curious to dive into the historical data from the Maltese region. Could you help me get an overview of the different variables we have for past studies?"

Importantly, convergence is not guaranteed: LLM Sim's active information need may never fully match the latent one. For reference, we display the prompt provided for LLM Sim in Figure 3.

You are simulating {domain_expert_desc}, who is interacting with a data discovery system to explore insights from an enterprise dataset.

{Depending on system:
- Pneuma-Seeker: The system represents your information need as a set of target schemas and SQL statements that, if executed, will provide the answer. It can combine, transform, and reason over data to assist your exploration.
- FTS or Pneuma-Retriever: The system only returns relevant tables based on your description. It does not infer your deeper intent, combine, or analyze data. }

Scenario:
- The system already has access to internal datasets.
- You (the simulated user) are familiar with the domain and have seen similar datasets before.
- You are not uploading new datasets or asking if they exist — you assume they do.

Possible eventual goal (unknown at start):
{question}

Behavior:
- Explore and refine your question step-by-step depending on the system's responses.
- Be vague or explore tangents, just as a curious analyst would.
- Only arrive at the specific question above if the system's output correctly leads you there.

Continue your role as the domain expert. This is the conversation so far (respond as if prompting the system directly):
YOU: {initial_broad_prompt}

Figure 3: Prompt for LLM_Sim

4.1 RQ1: Convergence

Convergence means the LLM Sim's active information need matches its latent information need. We introduce two metrics: **(1) percentage of convergence**, which is the proportion of benchmark questions for which LLM Sim converges, and **(2) median turns to convergence**, which is the median of the number of times LLM Sim has to prompt a system to achieve convergence (with an imposed limit of 15).

We compare Pneuma-Seeker with three baselines: BM25-based full-text search (FTS), Pneuma-Retriever, and LlamaIndex[2] (a representative RAG system). FTS and Pneuma-Retriever are static systems that only return tables, represented by their columns and sample rows. LlamaIndex adds an LLM on top of a top-k vector retriever to interpret the retrieved data for LLM Sim.

The results, as shown in Figure 4 and Figure 5, indicate that Pneuma-Seeker consistently achieves the highest percentage of convergence and similar median turns to convergence as LlamaIndex. Both Pneuma-Seeker and LlamaIndex not only surface relevant data but also interpret it, contextualizing their responses to LLM Sim. Even though the initial queries from LLM Sim are more general, the systems surface relevant data and suggestions and communicate them back to LLM Sim. Subsequently, LLM Sim responds to the systems to explore further and move closer to uncovering its latent information need, and the systems adjust accordingly (e.g., by modifying the state).

[2]https://www.llamaindex.ai/

On the other hand, both FTS and Pneuma-Retriever struggle to converge because LLM Sim has to interpret the results themselves. Additionally, even though Pneuma-Retriever can find the correct tables in almost all questions, LLM Sim can only observe sample rows to prevent hitting the context limit. Even with `GPT-4o`'s 128k context limit, 2-3 turns are enough to exceed the limit with our datasets. In most of the questions, LLM Sim keeps trying to adjust its queries to get more specific information from the retrieved tables.

There is a latency trade-off between Pneuma-Seeker and the other systems. On average, Pneuma-Seeker takes 70.26 seconds to respond to a prompt, while FTS and Pneuma-Retriever answer almost instantaneously. In addition, Pneuma-Seeker's LLM, `OpenAI's O4-mini`, incurs $1.1 and $4.4 for every 1 million input and output tokens, respectively. For reference, we include the estimated average costs of interactions between LLM Sim and Pneuma-Seeker across several models in Table 2.

4.2 RQ2: Accuracy

Converging to the latent information need is not enough; users ultimately want to get accurate answers for their information needs.

Table 2: Estimated Average Token Usage and Costs Across Different LLMs

Dataset	Avg. In Tokens	Avg. Out Tokens	Haiku 4.5		**O4-mini**		O3		gpt-5.1		Sonnet 4.5		Opus 4.5	
			In	Out	**In**	**Out**	In	Out	In	Out	In	Out	In	Out
Archeology	248,351	2,854	$0.25	$0.01	**$0.27**	**$0.01**	$0.50	$0.02	$0.31	$0.03	$1.49	$0.04	$1.24	$0.07
Environment	149,011	1,712	$0.15	$0.01	**$0.16**	**$0.01**	$0.30	$0.01	$0.19	$0.02	$0.45	$0.03	$0.75	$0.04

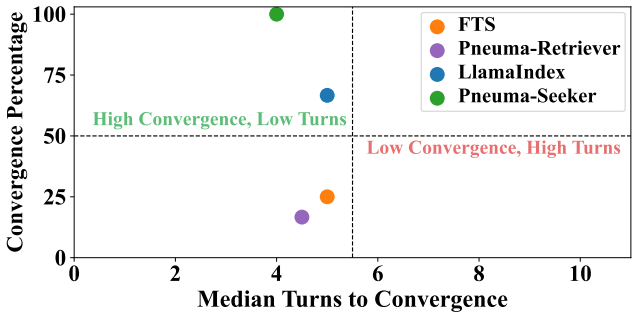

Figure 4: Comparison of Median Turns to Convergence vs. Convergence Percentage (Archeology Dataset)

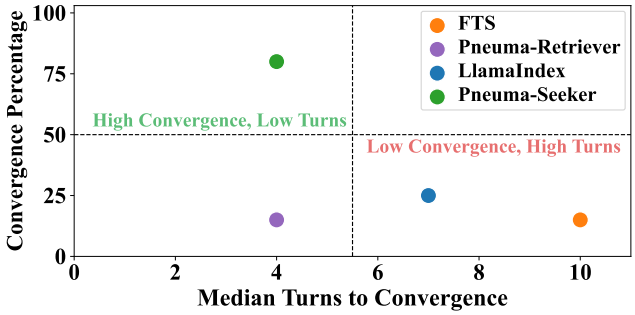

Figure 5: Comparison of Median Turns to Convergence vs. Convergence Percentage (Environment Dataset)

FTS and Pneuma-Retriever are not designed to provide answers, so we exclude them from RQ2 and include new baselines: DS-Guru and OpenAI's O3. DS-Guru is the Kramabench's reference framework, in which it instructs an LLM to decompose a question into a sequence of subtasks, reason through each step, and synthesize Python code implement the plan. We select the O3-based DS-Guru, as it is the best-performing one.

O3 is one of the best reasoning models right now with 200k context limit. For each benchmark question, we provide it with the whole relevant tables, so it has every necessary information to answer the question. However, we encountered context length exceeded errors with O3 in 6 out of 12 archaeology questions and 17 out of 20 environment questions. O3 answers none of the six archaeology questions correctly, but answers two environment questions correctly. Overall, passing all relevant context is still not a scalable approach.

For the remaining systems, the results are shown in Table 3. Pneuma-Seeker outperforms all systems across all datasets. Pneuma-Seeker outperforms DS-Guru, even though Pneuma-Seeker uses a smaller and more cost-efficient model. LlamaIndex, on the other hand, does not answer any questions correctly because the questions require actual computation (e.g., computing average of a certain column), not just interpretation of the top-k context.

Table 3: Comparison of Accuracy across Datasets

System	Archeology	Environment
LlamaIndex	0.00%	0.00%
DS-Guru (O3)	25.00%	19.60%
Pneuma-Seeker	**41.67%**	**55.00%**

5 DISCUSSION

In this section, we discuss lessons learned from building Pneuma-Seeker and explain our vision for the Pneuma project.

5.1 Lessons Learned.

We built Pneuma-Seeker over the course of the past year. We learned the following lessons that collectively motivated the architecture presented in this paper and directly informed the design of each component and their interactions.

Dynamic pipeline enables better adaptability. Our early prototypes followed a fixed processing sequence. We can certainly adapt this pipeline to include more actions/tools and hence cover more use cases. However, we realized that evolving such pipelines requires an increasing amount of effort. As soon as we encounter a use case that is not covered by the system, e.g., requiring new actions or interactivity, we are forced to do an extensive re-engineering.

This happened repeatedly. For example, when we had to determine which actions are required or optional, whether to skip or restart some actions, and when we had to integrate new actions without destabilizing the existing pipeline. A static approach *can* be adapted to handle these scenarios, but the engineering burden grows quickly as the system incorporates more and more actions and functionality.

Dynamic pipelines are not a silver bullet either: if a necessary capability is missing (e.g., semantic joins), the system will still fail. However, the key difference is that missing capabilities in a dynamic framework are *localized*: once implemented, they naturally slot into the system's action space. In contrast, adding the same capability to a static pipeline necessitates revisiting the entire pipeline.

This realization aligns with the broader spirit of AI-enabled systems: instead of hard-coding rigid sequences, the system should select actions dynamically based on evolving state, available tools, and accumulated domain knowledge. It also resonates with the declarative philosophy of data processing: specifying *what* is needed, not *how* to accomplish it.

Existing systems such as ReAcTable [39] and Chain-of-Table [32] also adopt dynamic orchestration in a more specific setting: assuming a conventional tabular QA setting where a single, specific table is known upfront. In our setting, the system may need multiple tables and non-tabular data to satisfy users' information needs. Overall, factors such as handling user feedback, maintaining iterative alignment of the state, and integrating heterogeneous data sources all push strongly against static pipelines. These observations collectively led us away from predefined pipelines and toward Conductor's flexible, state-driven dynamic orchestration.

Context specialization is essential. A key challenge in dynamic systems is the design of components themselves. We observed the importance of specializing context across different components (e.g., Conductor and Materializer). Beyond reducing the chance of hallucinations (since each component focuses narrowly on its role), specialization helps prevent exceeding context-window limits, which are more easily reached when the LLM is prompted with multiple different roles at once.

Looking ahead, although there is active research on extending context windows and future models likely will support larger windows, there is no guarantee that the model will effectively attend to (or meaningfully use) all its input tokens, even if they fit. Current models already have much longer context windows compared to those available just 3 years ago, but recent work still shows that LLMs' performance on tasks such as question answering degrades as context length grows, even when the relevant information is fully retrievable [6]. Theoretical analyses such as [5] also show that longer sequence length dilutes the model's ability to focus on specific tokens. Therefore, we argue that context specialization remains beneficial, especially as the system grows, e.g., with new actions and tools.

Schema-first representation guides alignment. Reifying information needs as a relational model provides a shared anchor that both the user and the system can collaboratively reason about. It gives users a concrete object to sanity-check the results, rather than relying purely on natural-language, user-facing messages. The state (T, Q) becomes a structure that can be iteratively refined, corrected, or extended, reducing miscommunication.

Surfacing intermediate (T, Q) states allows both users (and LLM Sim) to detect subtle misinterpretations early, before they propagate into later steps. Natural language alone is insufficient; users need visibility into the evolving data model to provide concrete, meaningful feedback. It also forces users to think concretely about what exactly they need. Designing effective ways to communicate this evolving state (and studying how users interpret and act on it) remains an important direction for future user studies.

5.2 Our Vision

We began the Pneuma project a year and a half ago, building on over five years of research in data discovery. Our first milestone was Pneuma-Retriever [1]. In this paper, we have presented the current state of the project, which has since grown to address a critical bottleneck in data-centric organizations: as LLMs become increasingly capable and embedded in data workflows, the main challenge shifts to helping users articulate what they want to get out of data. Our vision is a system that treats this articulation process as a first-class concern. This is reflected in our design through the reification of information needs as relational schemas tailored to the user's active intent and constructed on-the-fly.

We highlight an important emergent effect of Pneuma-Seeker's design. By prompting users to clearly express their goals, disclose assumptions, and externalize their mental models—a behavior increasingly familiar in LLM-driven interfaces—Pneuma-Seeker captures what is often tacit knowledge. Within organizations, this "tribal knowledge" represents a collective brain trust. If made searchable and persistent, it could fulfill the vision of internal data markets we proposed in earlier work [9]. Such markets would enable organizations to extract far greater value from their data assets, going well beyond the immediate gains in discovery and preparation addressed in this paper. Over the past several years, we have made progress toward that vision, but extracting tribal knowledge remains a hard-to-crack barrier. The Pneuma project is our latest and most focused attempt to lower that barrier and in doing so, to unlock latent value from organizational data.

REFERENCES

[1] Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. 2025. Pneuma: leveraging llms for tabular data representation and retrieval in an end-to-end system. *Proc. ACM Manag. Data*, 3, 3, Article 200, (June 2025), 28 pages. doi:10.1145/3725337.

[2] N.J. Belkin. 1980. Anomalous states of knowledge as a basis for information retrieval. *Canadian Journal of Information Science*, 5, 1, 133–143.

[3] Raul Castro Fernandez. 2025. What is the value of data? a theory and systematization. *ACM / IMS J. Data Sci.*, 2, 1, Article 3, (June 2025), 25 pages. doi:10.1145/3728476.

[4] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: a data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 1001–1012. doi:10.1109/ICDE.2018.00094.

[5] Shi Chen, Zhengjiang Lin, Yury Polyanskiy, and Philippe Rigollet. 2025. Critical attention scaling in long-context transformers. (2025). https://arxiv.org/abs/2510.05554 arXiv: 2510.05554 [cs.LG].

[6] Yufeng Du et al. 2025. Context length alone hurts LLM performance despite perfect retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, (Eds.) Association for Computational Linguistics, Suzhou, China, (Nov. 2025), 23281–23298. ISBN: 979-8-89176-335-7. doi:10.18653/v1/2025.findings-emnlp.1264.

[7] Raul Castro Fernandez. 2025. Data discovery is a socio-technical problem: the path from document identification and retrieval to data ecology. In *IEEE Computer Society Data Engineering Bulletin*. Preprint available at Semantic Scholar (CorpusID:281957015). https://api.semanticscholar.org/CorpusID:281957015.

[8] Raul Castro Fernandez, Aaron J. Elmore, Michael J. Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How large language models will disrupt data management. *Proc. VLDB Endow.*, 16, 11, (July 2023), 3302–3309. doi:10.14778/3611479.3611527.

[9] Raul Castro Fernandez, Pranav Subramaniam, and Michael J Franklin. [n. d.] Data market platforms: trading data assets to solve data problems. *Proceedings of the VLDB Endowment*, 13, 11.

[10] Charles R. Harris et al. 2020. Array programming with numpy. *Nature*, 585, 7825, (Sept. 2020), 357–362. doi:10.1038/s41586-020-2649-2.

[11] Jeffrey Heer and Ben Shneiderman. 2012. Interactive dynamics for visual analysis. *Communications of the ACM*, 55, 4, (Apr. 2012), 45–54. doi:10.1145/2133806.2133821.

[12] Nils Jahnke and Boris Otto. 2023. Data catalogs in the enterprise: applications and integration. *Datenbank Spektrum*, 23, 89–96. doi:10.1007/s13222-023-00445-2.

[13] Carol C. Kuhlthau. 1991. Inside the search process: information seeking from the user's perspective. *Journal of the American Society for Information Science*, 42, 5, 361–371. doi:10.1002/(SICI)1097-4571(199106)42:5<361::AID-ASI6>3.0.CO;2-#.

[14] Eugenie Lai et al. 2025. Kramabench: a benchmark for ai systems on data-to-insight pipelines over data lakes. *ArXiv*, abs/2506.06541. https://api.semanticscholar.org/CorpusID:279250249.

[15] Younghun Lee, Sungchul Kim, Ryan A. Rossi, Tong Yu, and Xiang Chen. 2024. Learning to reduce: towards improving performance of large language models on structured data. (2024). https://arxiv.org/abs/2407.02750 arXiv: 2407.02750 [cs.CL].

[16] Patrick Lewis et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (NIPS '20) Article 793. Curran Associates Inc., Vancouver, BC, Canada, 16 pages. ISBN: 9781713829546.

[17] Liyao Li et al. 2025. LongTableBench: benchmarking long-context table reasoning across real-world formats and domains. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, (Eds.) Association for Computational Linguistics, Suzhou, China, (Nov. 2025), 11927–11965. ISBN: 979-8-89176-335-7. doi:10.18653/v1/2025.findings-emnlp.638.

[18] Lei Liu, So Hasegawa, Shailaja Keyur Sampat, Maria Xenochristou, Wei-Peng Chen, Takashi Kato, Taisei Kakibuchi, and Tatsuya Asai. 2024. Autodw: automatic data wrangling leveraging large language models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (ASE '24). Association for Computing Machinery, Sacramento, CA, USA, 2041–2052. ISBN: 9798400712487. doi:10.1145/3691620.3695267.

[19] Tianyang Liu, Fei Wang, and Muhao Chen. 2024. Rethinking tabular data understanding with large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Kevin Duh, Helena Gomez, and Steven Bethard, (Eds.) Association for Computational Linguistics, Mexico City, Mexico, (June 2024), 450–482. doi:10.18653/v1/2024.naacl-long.26.

[20] Yuhan Liu, Michael JQ Zhang, and Eunsol Choi. 2025. User feedback in human-LLM dialogues: a lens to understand users but noisy as a learning signal. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, (Eds.) Association for Computational Linguistics, Suzhou, China, (Nov. 2025), 2666–2681. ISBN: 979-8-89176-332-6. doi:10.18653/v1/2025.emnlp-main.133.

[21] Yu. A. Malkov and D. A. Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. (2018). https://arxiv.org/abs/1603.09320 arXiv: 1603.09320 [cs.DS].

[22] Gary Marchionini. 2006. Exploratory search: from finding to understanding. *Communications of the ACM*, 49, 4, 41–46. doi:10.1145/1121949.1121979.

[23] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*. Stéfan van der Walt and Jarrod Millman, (Eds.), 56–61. doi:10.25080/Majora-92bf1922-00a.

[24] Sunghyun Park, Han Li, Ameen Patel, Sidharth Mudgal, Sungjin Lee, Young-Bum Kim, Spyros Matsoukas, and Ruhi Sarikaya. 2021. A scalable framework for learning from implicit user feedback to improve natural language understanding in large-scale conversational AI systems. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, (Eds.) Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, (Nov. 2021), 6054–6063. doi:10.18653/v1/2021.emnlp-main.489.

[25] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2024. Semantic operators: a declarative model for rich, ai-based data processing. In https://api.semanticscholar.org/CorpusID:271218837.

[26] Mark Raasveldt and Hannes Mühleisen. 2019. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data* (SIGMOD '19). Association for Computing Machinery, Amsterdam, Netherlands, 1981–1984. ISBN: 9781450356435. doi:10.1145/3299869.3320212.

[27] Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: bm25 and beyond. *Found. Trends Inf. Retr.*, 3, 4, (Apr. 2009), 333–389. doi:10.1561/1500000019.

[28] Tefko Saracevic. 1996. Relevance reconsidered. In *Proceedings of the 2nd International Conference on Conceptions of Library and Information Science (CoLIS2)*. Peter Ingwersen and Niels Ole Pors, (Eds.) Royal School of Librarianship. Copenhagen, Denmark, 201–218.

[29] Murray Shanahan, Kyle McDonell, and Laria Reynolds. 2023. Role play with large language models. *Nature*, 623, 7987, (Nov. 2023), 493–498. doi:10.1038/s41586-023-06647-8.

[30] John W. Tukey. 1977. *Exploratory Data Analysis*. Addison-Wesley, Reading, MA. ISBN: 0-201-07616-0.

[31] Matthias Urban, Jialin Ding, David Kernert, Kapil Vaidya, and Tim Kraska. 2025. Utilizing past user feedback for more accurate text-to-sql. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics* (HILDA '25) Article 10. Association for Computing Machinery, Intercontinental Berlin, Berlin, Germany, 7 pages. ISBN: 9798400719592. doi:10.1145/3736733.3736739.

[32] Zilong Wang et al. 2024. Chain-of-table: evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=4L0xnS4GQM.

[33] Ryen W. White and Resa A. Roth. 2009. *Exploratory Search: Beyond the Query-Response Paradigm. Synthesis Lectures on Information Concepts, Retrieval, and Services*. Morgan & Claypool Publishers. ISBN: 978-1-59829-783-6. doi:10.2200/S00174ED1V01Y200901ICR003.

[34] Ryen W. White and Resa A. Roth. 2009. Exploratory search: beyond the query-response paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1, 1, 1–98. doi:10.2200/S00174ED1V01Y200901ICR003.

[35] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

[36] Amy X. Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *Proc. ACM Hum.-Comput. Interact.*, 4, CSCW1, Article 22, (May 2020), 23 pages. doi:10.1145/3392826.

[37] Shiqi Zhang, Xinbei Ma, Zouying Cao, Zhuosheng Zhang, and Hai Zhao. 2025. Plan-over-graph: towards parallelable llm agent schedule. (2025). https://arxiv.org/abs/2502.14563 arXiv: 2502.14563 [cs.AI].

[38] Xiaokang Zhang et al. 2025. TableLLM: enabling tabular data manipulation by LLMs in real office usage scenarios. In *Findings of the Association for Computational Linguistics: ACL 2025*. Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, (Eds.) Association for Computational Linguistics, Vienna, Austria, (July 2025), 10315–10344. ISBN: 979-8-89176-256-5. doi:10.18653/v1/2025.findings-acl.538.

[39] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024. Reactable: enhancing react for table question answering. *Proc. VLDB Endow.*, 17, 8, (Apr. 2024), 1981–1994. doi:10.14778/3659437.3659452.

[40] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. Db-gpt: large language model meets database. *Data Science and Engineering*, 9, 1, (Mar. 2024), 102–111. doi:10.1007/s41019-023-00235-6.

[41] Dongsheng Zhu, Weixian Shi, Zhengliang Shi, Zhaochun Ren, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. 2025. Divide-then-aggregate: an efficient tool learning method via parallel tool invocation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, (Eds.) Association for Computational Linguistics, Vienna, Austria, (July 2025), 28859–28875. ISBN: 979-8-89176-251-0. doi:10.18653/v1/2025.acl-long.1401.