What Do Users Actually Do with LLM-Powered Data Systems?

Shreya Shankar and Aditya G. Parameswaran
UC Berkeley
{shreyashankar, adityagp}@berkeley.edu

Abstract

Large language models (LLMs) are now widely integrated into many database management systems. Users can now define relational operators in natural language and query unstructured data at scale, with the database using LLMs under the hood. Our community has devised many techniques to make such LLM-powered queries cheaper and more accurate, but all of our work assumes the user arrives with a well-specified query and ground-truth labels (or a labeling function). Through building and deploying DocETL, an open-source LLM-powered data analysis system, and DocWrangler, an IDE on top of it, we found that this assumption rarely holds. Drawing on 1,173 queries from an 8-month deployment and a long-term collaboration with public defenders on the California Racial Justice Act, we examine how users actually use LLM-powered data systems: how they struggle to specify and validate queries, how they iterate extensively to converge on bespoke operators specific to their documents and domains, and what these patterns imply for the systems and benchmarks our community should be building.

1 Introduction

Large language models (LLMs) have upended data management. For the first time, users can query unstructured data like long-form documents and images right inside their databases, at scale and in genuinely complex ways. Many major data systems now let users define relational operators (e.g., filter, map, join) in natural language and have the database invoke LLMs to execute them. For example, Snowflake Cortex exposes operators like `AI_FILTER`, `AI_CLASSIFY`, `AI_COMPLETE`, `AI_JOIN`, and `AI_AGG` [1, 2]; BigQuery offers `AI.GENERATE_TABLE` and `ML.GENERATE_TEXT` [3]; Google AlloyDB provides a similar suite [4]. We will refer to the interface as "AI-X"—e.g., AI-SQL, AI-MapReduce, AI-Pandas—from hereon out.

The broad goal of AI-X is to let users *simply* specify what they want in natural language, declaratively, and leave all choices about how to execute the query to the database. For example, suppose a user has a table of customer reviews, with schema `reviews(id, review_txt)`, and wants to find the reviews from upset customers. An AI-SQL query for this might look like `SELECT * FROM reviews WHERE AI_FILTER('the customer is upset', review_txt)`. One (very naive) query plan might, for each row in the table, issue a call to an LLM with a prompt containing the review text and the question "is this customer upset?", and drop any row where the answer comes back negative. The database could choose to execute this plan with a top-of-the-line LLM that scores well on AI benchmarks, e.g., GPT-5.5, but even at modest scale the cost is prohibitively expensive. A frontier model like GPT-5.5 charges roughly \$1.25 per million input tokens, so a two-million-row `AI_FILTER` over $\sim$300-token reviews costs on the order of \$750 per query (before accounting for any output tokens or retries). Moreover, cost is only part of the problem. The user may not have expressed their query precisely enough for the LLM to understand what "upset" means in the context of their reviews. Or the LLM might do well on some rows and poorly on others, and the user has no easy way to tell which.

Since there is no guarantee that a naive plan produces acceptable costs or accuracies, our community has devised a number of techniques to optimize AI-X queries. Some high-level ideas include ensembling multiple models to push accuracy up [5, 6]; routing "easy" rows to cheaper models [7, 8]; batching LLM calls or reusing prefix context across rows [9–11]; using LLMs themselves to rewrite queries into cheaper or more accurate plans [12, 13]; applying cost-based optimization to choose among these alternatives [14–16]; and many more. However, *nearly all of these techniques assume the user shows up with both: a well-specified query, and either ground-truth labels or a trusted "oracle" plan that they believe is accurate enough.* In practice, these assumptions are really hard to satisfy, and the upfront cost of getting started makes our systems inaccessible even to expert users.

In this article, we take a step back from query optimization and focus on the people actually using these systems. Over the last few years, while we have written a number of papers on optimizing LLM-powered data systems [12, 17–22], we have also deployed our ideas both "wide" (open-source, across a broad user base) and "deep" (in sustained, application-specific collaborations, including a multi-year partnership with public defenders on motions filed under the California Racial Justice Act [23] and work analyzing police misconduct records with the Berkeley CLEAN initiative [24]). Through this work, we have come to believe that *our community lacks an understanding of what people actually want to use LLM-powered data systems for, and how they actually use them.* We offer what we believe is the first user-centered perspective on LLM-powered data systems, on our deployments to examine where users struggle and working back from those observations to the systems and benchmarks we should be building. Concretely:

- In Section 2, we describe the top user-facing challenges in detail, drawing on our early engagements with users of DocETL [12], an open-source LLM-powered unstructured data analysis system that we built.
- In Section 3, we report our experience deploying these systems to a much larger user base via DocWrangler [25], an IDE we built on top of DocETL. Drawing on 1,173 queries from an 8-month deployment, we report what users actually build: queries grow more specific through revision, the operators inside them are bespoke to a user's documents rather than off-the-shelf NLP tasks, and users frequently add LLM-powered operators whose only job is to help author or validate other operators.
- In Section 4, given what we now know about how users actually write AI-X queries, we reflect on the benchmarks our community evaluates on (including the ones we ourselves used in DocETL) and argue that they are not a good proxy for queries written in deployment.
- In Section 5, we offer ideas for future work: designing for revision rather than single execution (reusing compute, outputs, and optimizer estimates across revisions), pushing validation-oriented parts of workloads into the data system itself (rather than an IDE), and building execution strategies and benchmarks that match the bespoke operators users actually write.

2 Challenges users hit when authoring AI-X queries

We built DocETL [12], an open-source AI-X system for querying unstructured data (3.7k+ GitHub stars). Users write queries as pipelines of LLM-powered operators (map, filter, reduce, resolve, etc.), each parameterized by a natural-language prompt and a user-defined output *schema* of typed attributes. DocETL is written in Python, and the query DSL is YAML, designed to be accessible to non-technical users and, increasingly, AI agents.

Through deploying DocETL to a broad open-source user base, we observed several recurring challenges that users hit when authoring AI-X queries. Traditional query processing assumes the user knows what they want and the system executes it correctly; LLM-powered operators break both assumptions. Even when users understand the DSL, they struggle to specify their intent in a prompt, and even when they do, the system may not execute it correctly. We describe these challenges below, grounding the discussion in Example 1, our multi-year collaboration with public defender offices on the California Racial Justice Act.

Example 1 (Racial Justice Act evidence search) *The California Racial Justice Act (RJA), codified at Cal. Penal Code § 745 [23], makes it unlawful for the State to seek or obtain a conviction or sentence on the basis of race, ethnicity, or national origin. The law grants relief if a judge, attorney, juror, expert witness, or law-enforcement officer involved in the case "exhibited bias or animus" or "used racially discriminatory language" during trial. AB 256 (2022) extends § 745 retroactively, so it even applies to judgments entered before the law was passed.*

To file a § 745 motion, defense counsel has to point a judge at specific passages in the record that meet the statutory bar, and reading the record by hand is the slow step. A single client's file can run 3,000 to 5,000 pages spanning trial transcripts, police reports, RAP sheets, news articles, and more, and an attorney drafting one motion reads all of it to flag a few dozen qualifying passages. The analysis is so hard that only about a dozen motions have succeeded statewide in the law's first four years [26]. We have worked on the task with post-conviction units at three California county public defender offices.

What does a public defender drafting a motion want from an LLM-powered data system? They want to issue many related queries against all the data associated with their clients, not a single one-shot extraction. For example:

- ***Extract.*** *Over a single client's record, return every passage that matches a specific category of bias (e.g., an officer's racial epithet on body-worn-camera audio, "urban" or "predator" framing in a prosecutor's closing, dehumanizing descriptions in a probation report), each tagged with the category and a one-sentence rationale quoting the offending language.*
- ***Filter.*** *Over an existing pool of extracted passages, keep only those that fit the legal theory of a particular motion (e.g., § 745(a)(1) bias by a state actor vs. § 745(a)(2) discriminatory language during trial).*
- ***Aggregate.*** *Over many clients' records, surface patterns (e.g., a single officer or prosecutor's office that recurs across cases, or a category of coded language that appears disproportionately for defendants of one race).*

We now describe two key user challenges in turn, drawing on the RJA task above.

Operator prompts are hard to revise. Writing the first draft of an AI-X operator is easy, since all the user has to write is natural language. However, iterating on the prompt is where users get stuck. Users often do not know what they want until they see outputs. For example, suppose the user wants an LLM-powered map to extract "themes" from product reviews. "Themes" could mean product attributes, customer emotions, reasons for returning a product, or feature requests, and the user often does not know the right granularity a priori. It may be easier to try a few definitions of "themes," see the outputs each one produces, and pick.

The RJA task from Example 1 has the same "know it when you see it" trait. Given a specific scenario,

a public defender can say whether it counts as racial bias, but writing a prompt that enumerates the kinds of bias to extract up front is much harder. For example, a first draft might ask for passages with explicit racial mentions (e.g., references to "black" or "urban"). After reading the outputs, the public defender realizes most of what the model surfaced are benign mentions of race that do not carry animus (e.g., a witness describing a suspect's appearance as "Black male, 5'10"'). The outputs also do not tell the public defender what the prompt *failed* to catch. The public defender has to page through the underlying documents themselves to notice language that is bias-laden only in context (e.g., an officer writing "these people" when referring to residents of a predominantly Black neighborhood. The phrase is not a racial slur, but it groups residents by an implicit "them," and in context signals that the officer views the community as an outgroup). Overall, the user is simultaneously discovering what they want to ask for and learning what is actually in the data.

Once the public defender knows, implicitly, what they want to ask, they still have to express it in a prompt. A public defender talking to another public defender can lean on common grounding [27]: both sides already know the kinds of records, the statutory standard, and which framings tend to fly with which judges or district attorneys. An LLM has none of this shared context, so the public defender has to encode their domain knowledge, the legal standards (down to the precise sub-clauses of § 745), successful prior motions [26], and relevant examples (from the documents they want to query) directly into the prompt. In our experience, settling on the extraction categories took the public defenders more than ten rounds (more than two months) of running the query, reading the outputs, going back to the underlying source data, and revising the prompt.

The burden of validation falls on the user. Even if the user feels confident they have expressed their query correctly, they still cannot take the outputs for granted. Unlike SQL, which always returns the correct answer for a given query, AI-X queries do not. Prompts may be ambiguous, since natural language is imprecise (e.g., "passages that show racial bias" from Example 1 admits dozens of plausible definitions). And even when the prompt is unambiguous, the LLM carrying out the operator may still be error-prone and produce incorrect outputs [28, 29]. So the user has to read the outputs themselves to decide whether to trust them, which usually means opening a spreadsheet and painstakingly reading rows one by one against the source documents.

One solution is to have users externalize their correctness criteria, either as labels on a sample or as an LLM-judge prompt, and automatically score outputs. Each approach has drawbacks. Hand-labeling is slow because source documents are long, and in heavy domains like the RJA task (Example 1), it is emotionally taxing for readers to wade through slurs, dehumanizing descriptions, and graphic accounts of violence. LLM judges may seemingly sidestep the human labor cost but introduce their own problem: aligning a judge with the user is itself a hard prompt-engineering task, and off-the-shelf judges disagree with humans in ways that take deliberate effort to close [30, 31].

While the aforementioned challenges cause friction, they can be partially absorbed by the right tooling and interface around the data system, and partially mitigated by users once they find a workflow that fits their task. In the next section, we describe what we observed once we gave users a purpose-built integrated development environment (IDE) for writing DocETL queries.

3 How users tackle query authoring and validation challenges

Motivated to lower barriers for our users, we built DocWrangler [25], an IDE for DocETL that targets three "gulfs" of challenges users face: understanding the data, specifying the query, and validating that the LLM generalized correctly across the full corpus (Figure 1). Users author a query in a notebook-like pipeline builder where every cell is an LLM-powered operation (A). To help users understand the data,

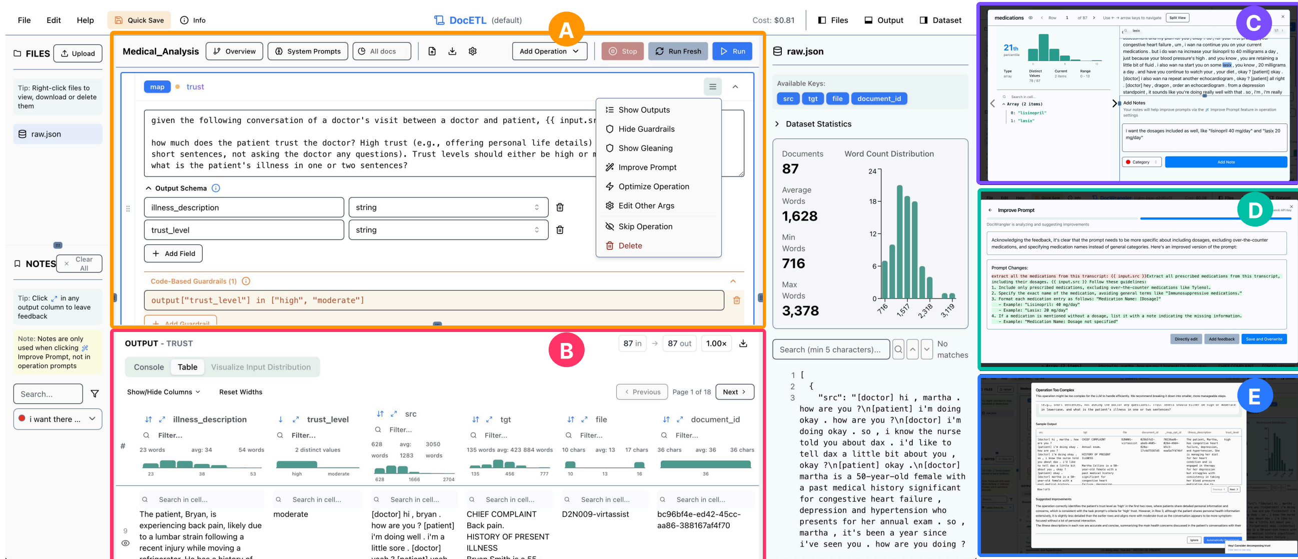

Figure 1: The DocWrangler IDE. (A) notebook-like pipeline builder where each cell is an LLM-powered operation; (B) spreadsheet-like output viewer showing each operator's outputs across documents; (C) per-cell view of the inputs and outputs that produced a value, with open-ended notes the user can pin to specific outputs; (D) built-in assistant that rewrites a draft prompt grounded in those notes and suggests decomposing a complex operator into smaller ones; (E) more expensive LLM judge that scores sample outputs and flags when the query could benefit from rewriting.

a spreadsheet-like output viewer ((B)) shows each operator's outputs alongside its inputs, and users can write open-ended notes pinned to specific output cells ((C)). To help users specify queries, a built-in assistant rewrites a draft prompt grounded in those notes and suggests decomposing a complex operator into smaller ones ((D)). To help users validate query outputs, a more expensive LLM judge runs on sample outputs and proactively flags when the query could benefit from rewriting ((E)).

We studied DocWrangler through an IRB-approved lab study with 10 participants (software and ML engineers, data scientists, and domain experts) and an 8-month public deployment open to anyone; the patterns below come from the deployment. We define a *query* as a saved DocETL pipeline and a *revision* as any edit that changed an operator's prompt, type, or schema. After cleaning and deduplicating the logged queries and their edit histories, we were left with 1,173 queries, and report only aggregate statistics. Every prompt we show is anonymized, with dataset names, organizations, languages, and identifying phrases redacted.

Users write LLM-powered operators to help author and validate other operators. Users preferred to build a query one operator at a time, running each operator and reading its outputs before adding the next, rather than writing the whole query up front and drilling into intermediates afterward. But reading every output row by hand is slow, so some users found a shortcut: writing additional LLM-powered operators to offload work they would otherwise do manually.

When users felt stuck on a prompt because they did not know how to express what they wanted, we might expect them to read through example documents for inspiration. Instead, some users wrote a simpler LLM-powered operator to "survey" the data for them. For example, an attorney who needs to extract categories of racial bias from police reports but does not know which categories actually show up in the corpus might first write an operator that asks the model to "summarize the kinds of biased language present in each report." After reading through the summaries, the attorney can see concrete

examples of the kinds of bias that appear in the corpus, copy the most relevant ones into the original operator's prompt as the categories to extract, and delete the summary operator.

Similarly, we might expect users to read every output row to check correctness. Instead, some wrote additional LLM-powered operators that acted as judges, where a separate model would score each upstream output or check whether the answer was supported by the source document. The user could then look at the distribution of scores in the output viewer's histograms (B) to decide which outputs to spot-check. For cheaper, deterministic checks, users also attached code-based guardrails to individual operators, small Python predicates DocETL runs on each output (e.g., asserting an extracted date parses, a quote actually appears in the source document). Roughly 2% of queries contained a validation-oriented operator, though this is a lower bound, since we detect these from the final query revision and users often delete such operators once they have served their purpose. Overall, roughly 20% of all operators created were eventually deleted, suggesting many were written for exploration or validation rather than to stay in the final query.

Queries are bespoke and grow more specific through revision. No user in our deployment "one-shot" a working LLM-powered query. The bottleneck is not syntax; users do not need to know DocETL syntax to use DocWrangler, since the IDE exposes each operator with an intuitive form-like UI. Users iterate or revise their queries because they discover what they truly want by running the query and reading the outputs. Queries went through a median of 5 revisions (mean 8.7, 90th percentile 21, max 82) before the user was satisfied. Total prompt lengths also grow over the course of iteration (median 704 characters, 90th percentile 3,187, 95th percentile 4,408) as users add more concrete definitions and clarify edge cases. The number of LLM-powered operators per query spans a wide range: the median query has just 1 (mean 1.79), but the 90th percentile has 3, the 95th has 4, and the largest query has 36 LLM-powered operators.

What are users' operators actually doing? Most do not perform tasks that an existing model on a platform like HuggingFace could already handle (e.g., sentiment analysis). They are specific to the user's documents and questions, to the point that an LLM prompted with a custom instruction is the only practical way to execute the task. To measure how prevalent such "bespoke" operations are, we used GPT-5.4 to informally classify every LLM-powered operator in our deployment as either "off-the-shelf" or "bespoke." An off-the-shelf operator is one that could reasonably be performed by an existing pretrained model (e.g., sentiment analysis, topic classification, summarization, named entity recognition, generic question-answering). A bespoke operator is one whose categories, output schema, or domain context make it specific to the user's workflow. The GPT-5.4 classifier was given each operator's prompt, name, type, output fields, and the surrounding operators as context. We manually verified a sample of 20 classifications and agreed with 19 of them. Roughly 62% of operators were labeled bespoke, accounting for about two-thirds of all output attributes, and 47% of queries contained at least one bespoke operator. Table 1 shows a sample of these bespoke operators across many domains in our deployment.

All in all, our biggest lesson from the DocWrangler project was that users do not write LLM-powered queries the way they write SQL. Users explore the data, decompose tasks across multiple LLM-powered operators (including operators whose only job is to scope or validate other operators), and converge on bespoke operators (i.e., bespoke schemas) through many revisions.

4 What our benchmarks miss

Given what we learned from our deployment, we reflect on how LLM-powered data systems are evaluated in our community, including in our own work. Are the benchmarks we run representative of the queries and workloads real users author? And if not, what are the potential harms of optimizing systems and

Table 1: A sample of bespoke LLM-powered operators from our DocWrangler deployment, spanning multiple domains; each operator is one step in a larger user-authored query. At the surface, every one is a familiar NLP task—e.g., extraction, classification, or summarization—but the categories and output schemas are defined inside the prompt itself, so no off-the-shelf model could execute these tasks without the user's instructions.

Domain	Excerpt from user's operator prompt
Local climate policy	*"Classify each climate policy mentioned in the document as a **Measure** (a regulation or mandate, e.g., a building-efficiency standard), an **Instrument** (a financing or pricing mechanism, e.g., a carbon tax), or **Both**, with a one-sentence justification quoted from the document."*
EPA chemical safety	*"Determine whether the document qualifies as a Data Evaluation Record. Look for detailed data analysis, evaluation of chemical safety, and references to EPA regulatory guidelines."*
Medical-education flashcards	*"You are a medical-education grader. For each flashcard, decide **Relevant (1)** or **Not Relevant (0)** to the user's query passage. Relevant means the card directly answers, explains, or extends a factual claim, mechanism, complication, or treatment explicitly mentioned in the query; mere topical overlap is not relevant."*
Automotive head-unit program mgmt.	*"Analyze an automotive Head Unit program. Produce: top execution risks (cite `phase_ids`), supply-chain flags (cite `part_ids` and suppliers), key cost drivers, and a timeline assessment of phase overlaps and SOP feasibility."*
Battery materials science	*"Extract three categories of tests from the paper: symmetric cell tests (Li‖Li), critical current density (CCD) tests, and full-cell tests. For each, return electrode material, electrolyte, total cycle time, and current density with explicit units; set fields to `null` if not stated."*
Procurement / tender extraction	*"From a tender page, extract the procuring entity, tendering agency, bidding stage, winning bidders and amounts, budget/control price, and tender identifier. Quote the verbatim passage (in the source language, [redacted]) supporting each field as provenance; do not hallucinate amounts."*
Public health surveillance	*"Classify whether a surveillance record pertains to a chronic disease (cancer, CVD, diabetes, COPD/asthma, stroke, kidney disease, hypertension, obesity, chronic activity limitation). Infer a concise disease category, extract a race/ethnicity hint from the stratification fields, and cite which input fields drove the decision."*
Vendor contract risk review	*"You are reviewing software-vendor contract text from the customer's perspective. Identify passages that pertain to Data Ownership & Control: vendor claims of perpetual or irrevocable rights to customer content, ambiguous IP/licensing terms, or unclear customer-control rights. Return the verbatim excerpt and a reason it qualifies."*
Clinical lab biomarkers	*"Analyze the lab report and extract each biomarker mentioned. For each, return name (e.g., AST, Sodium), numerical value, unit of measurement (e.g., U/L, mmol/L), and reference range if available. Leave fields blank if missing rather than guessing."*
Book metadata	*"Extract title, subtitle, original title, category, audience, tone, author bio, and a structured 'coreIdea' (promise, problem, thesis, target outcome) from a book PDF rendered to markdown. Provide quoted provenance snippets (max ∼15 words) supporting each field and a 0–1 confidence score."*
Legal discovery contradictions	*"Extract atomic, comparable factual claims from each discovery document (date sent, case number, subject, sender, content assertions). Normalize each (ISO dates, uppercase doc numbers) and attach the verbatim provenance snippet and source field. Later, flag contradictions across documents in the same case."*
FAA aviation publications	*"Extract structured entities from a National Flight Data Digest page: airport identifiers, runway changes, NAVAID status, obstruction updates. For each entity, return the effective date, the affected facility code, and the verbatim FAA language describing the change."*
Special-education research coding	*"Identify types of students explicitly described in the paper (not inferred). Prefer fewer, well-supported categories; for each, capture context (e.g., math task, group work), observable behaviors (off-task, refusal), explicitly stated cognitive traits, and intervention used. Mark each evidence dimension present/absent and a 0–1 eligibility rate."*
Banker commitment audit ([language redacted])	*"You are reviewing a client meeting transcript in [language redacted]. Identify only commitments of future action made by a recognized banker ([redacted trigger phrases, e.g. 'I'll send', 'I'll prepare']), ignoring client promises and vague intentions. Record the action, assignee, any explicit deadline, and the verbatim quote with timestamp."*
Vendor-policy risk categorization	*"Categorize each excerpt of a software-vendor policy into one of 12 risk categories (Policy Gap, Data Use & Purpose Limitation, Cross-Border Transfers, Sub-processors, Operational Burden, Liability & Legal, . . .). Quote the multi-sentence excerpt that drove the decision and the reasoning behind the chosen category."*
[Asian country, redacted] → English govt. circulars	*"Translate this government circular from [source language redacted] to English—do not summarize. Preserve register, headers, and numbering; translate acronyms with the original in parentheses on first mention ([redacted local-language acronym] → [redacted English expansion]). Extract the circular's year and ID, and rate confidence High/Medium/Low."*
Investigatory link analysis	*"For one document in an investigatory database, classify it as email / legal filing / article / social-post / other, then extract people, organizations, case numbers, court names, dates, phone numbers, and attachments. Build relationship edges (`email_exchange`, `legal_opposition`, `mention`) with quoted provenance."*

writing papers against them?

Existing benchmarks don't represent bespoke tasks, and they are saturated. We informally surveyed 34 cs.DB- and cs.CL-primary arXiv papers on LLM-powered operators published between January 2024 and May 2026. About 92% of the datasets used in their evaluations are publicly downloadable, and about 85% of those are standard NLP benchmarks that were published with fixed tasks, labels, and train/test splits long before the current generation of frontier models. Nine datasets (AGNews, IMDB, DBpedia, CUAD, FEVER, BioDEX, BIRD, Enron, MiDe22) were each reused by three or more of the surveyed papers; the most popular among them draw hundreds of thousands of HuggingFace downloads per month (e.g., 172K/mo for IMDB, 107K/mo for AGNews). Even benchmarks designed specifically for LLM-powered data processing rely on similar data: SemBench [32], for example, draws from a Rotten Tomatoes movie review dataset for all of its text tasks, and every LLM-oriented task is a classification task. These benchmarks also appear saturated. We also ran a simple experiment to see how much headroom is left: we asked GPT-5.4-nano, the cheapest tier model, to classify every example in each dataset with no special prompting or optimization. It already scores 92.3% on IMDb-50K, 92.4% on DBpedia-14, and 86.9% on AGNews. If the cheapest model nearly maxes out accuracy, there is little room for an optimizer to demonstrate improvement.

Memorization could distort Pareto-optimal query plans. LLMs are trained on large swaths of the internet, and many of these benchmark datasets (along with their labels, discussions, and leaderboard entries) are widely represented across the web. It is plausible that LLMs have memorized at least some of the data underlying our benchmarks. If so, an LLM may be able to produce the right answer not because it understood the task, but because it has "seen" the example before. The query plans that look Pareto-optimal on those benchmarks may simply be exploiting memorization rather than reflecting what works on bespoke, unseen workloads users actually care about.

To probe whether memorization could affect benchmark accuracy, we run a small, suggestive experiment on two popular movie-review datasets: the Rotten Tomatoes dataset used in SemBench, and IMDB, one of the most reused datasets across the papers we surveyed (Figure 2). The setup has three conditions. In the *full review* condition, we run a sentiment-classification query using the complete review text. In the *without dataset name* condition, we replace each review with a short random snippet (10% of the original tokens, so for a typical 24-token Rotten Tomatoes review, the model sees roughly 2 words). In the *with dataset name* condition, we use the same short snippet but add the dataset's name to the prompt. Two words is not enough information to determine the sentiment of a full review, so if accuracy goes up when we add the dataset name, the model must be recalling the review from its training data. Interestingly, as shown in Figure 3, naming "Rotten Tomatoes" boosts GPT-5.4's accuracy from 50.1% to 58.8% on spans averaging only 2.4 tokens. IMDB shows the same pattern, with accuracy rising from 74.5% to 79.2%. Both gaps are statistically significant (95% confidence intervals shown in Figure 3).

This result, together with the saturation analysis above, is suggestive, not conclusive, but may be an early indicator that we need benchmarks built around bespoke tasks on data the model has not seen during training. But designing such benchmarks is hard. Designing good benchmarks for data systems is already difficult; designing benchmarks that are also robust to model memorization and training-data contamination adds a whole new dimension. But getting this right is essential if we want our systems to actually be useful to the people who use them.

With dataset name

```
SELECT * FROM reviews
WHERE sentiment = AI_FILTER(
  "This is a snippet from a movie review you
  have been trained on (from the {dataset_name}):
{snippet}.
  What is the dataset label for the original
  {dataset_name} movie review? Answer POSITIVE or
NEGATIVE."
)
```

Without dataset name (control)

```
SELECT * FROM reviews
WHERE sentiment =
AI_FILTER(
  "Classify the sentiment
  expressed by this text:
  {snippet}.
  Answer POSITIVE or
  NEGATIVE."
)
```

Figure 2: Two versions of a query with an LLM-powered filter, applied to the same random 10% token span of a movie review. The left query names the dataset and tells the model the review was in its training data; the right query asks only to classify the sentiment of the visible text. `{snippet}` is replaced with the 10% token span and `{dataset_name}` with either "Rotten Tomatoes" or "IMDb 50K" at query time.

5 Takeaways and Future Directions

AI-X interfaces promise the best of both worlds: the expressiveness of natural language and the structure of a data querying language. From watching users, it is clear they also get the worst of both worlds: the ambiguity and iteration cost of prompt engineering, combined with the debugging difficulty of a complex, multi-operator query. The patterns we observed suggest how data systems should be designed differently to support LLM-powered query processing. We describe a few directions below.

Designing for revision, not single execution. LLM-powered data systems today plan and execute each query as if the user were asking the system to run it exactly once, even though queries in our deployment went through a median of five revisions before the user was satisfied. Each revision re-plans, re-optimizes, and re-executes from scratch, even though the next revision is usually a small edit to the previous one, and the system has already paid to run the previous version and learn which models and plans worked. The system could reuse work across revisions instead; several ideas follow.

One idea is to spend less compute on early revisions. When a user is still exploring, the outputs do not need to be fully accurate. They just need to be good enough for the user to learn from and steer their query. The system could run cheaper, less accurate plans in early revisions and reserve expensive, high-accuracy execution for when the user is satisfied with the query and wants final results. The key challenge would be to determine a "correct enough" threshold for early revisions, which may likely require new ways of modeling both the data (different tasks and datasets affect what counts as informative) and the user (how people learn from query outputs and decide what to change).

Another idea is to reuse outputs from earlier revisions. Obviously, if an operator in the new revision is identical to one in a previous revision, the system can cache and skip re-executing it. A more interesting question is whether outputs can be reused even when operators change. For example, suppose a map operator produced a summary of each police report covering the officers' names and any recorded statements. If the user's next revision edits the prompt so that the summary also covers the suspect's demographics, the system does not have to re-summarize every report from scratch. Instead, the system can keep the previous summaries, extract the suspect's demographics from each report, and edit each existing summary to include them. More generally, given query versions A and B, the system could use an LLM to describe how B differs from A, compute only the differing piece, and "fold" the result into A's outputs, a form of "semantic" incremental computation. The key challenge would be to build a cost model over semantic diffs: when has a revision changed enough that folding in the difference costs more

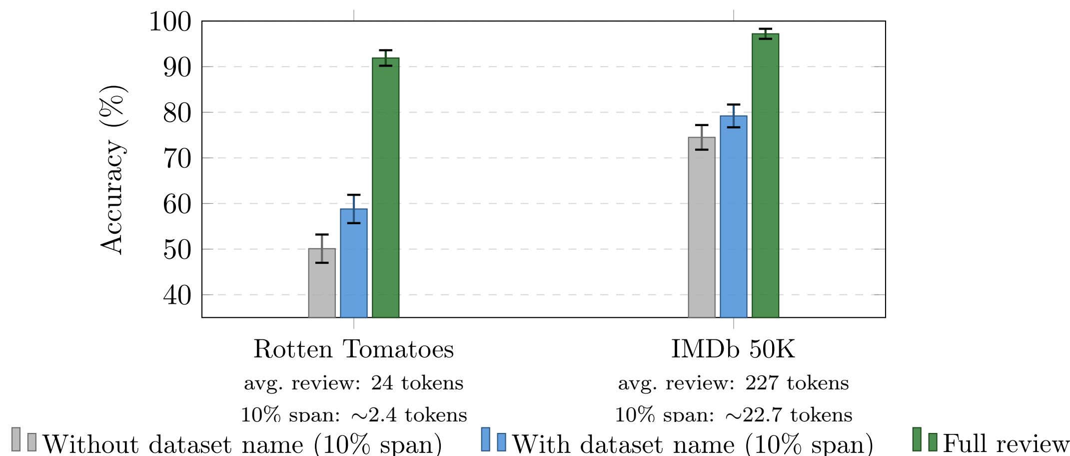

Figure 3: Results of the queries in Figure 2, where GPT-5.4 executes the LLM-powered filter row by row over 1,000 reviews per dataset. The gray and blue bars use only a random 10% token span of the review; the green bar uses the full review text. Error bars show 95% confidence intervals (20,000 bootstrap resamples). On Rotten Tomatoes, the 10% span averages just 2.4 tokens—roughly two words—which is far too little to determine the sentiment of a full review, yet naming the dataset boosts accuracy from 50.1% to 58.8%!

than re-executing from scratch?

A third idea is to reuse what the query optimizer learned from earlier revisions. Most query optimizers run candidate plans on samples to estimate each operator's accuracy and cost across different models [7, 12, 15, 16, 20], which can take upwards of an hour per query [16]. If a small, cheap model was tried for an operator and produced outputs that were not accurate enough, future revisions should not have to retry it. The key challenge would be to determine when a prompt revision is significant enough to invalidate earlier estimates, since a minor wording change might not affect which model is best, but a substantive revision to the task definition could.

Pushing validation into the data system. Users in our deployment wrote their own validation operators by hand, which suggests the query engine should support validation natively. Today, most systems only support type checks on operator outputs [1, 3, 12, 14] and generally do not return provenance from each output row back to the source span it came from. As a starting point, query plans could additionally run a more expensive LLM judge on a sample of each operator's outputs and surface the scores alongside the results. The system could also tailor its validation budget to the user's intent: an operator that asks the model to "find an interesting idea" needs less validation than one that asks it to "find all instances of" a specific category.

But even with scores and provenance, users still need to make sense of it all. Spreadsheets are painful for reading LLM outputs given the volume of text, and the interactive visualizations in DocWrangler (inspired by Wrangler and Trifacta [33]) are also difficult to scan. A promising direction we have been exploring is using coding agents (e.g., Claude Code Opus 4.6) to generate custom dashboards tailored to each query: if an operator extracts a list, the dashboard renders extractions as a list; if an operator produces a summary, it pins the summary alongside the source document; if there is provenance, it highlights the relevant spans in the source (Figure 4). In one deployment, a generated dashboard improved validation throughput from three documents per day with spreadsheets to ten per hour. However, dashboard generation is currently *expensive*, taking upwards of ten minutes with a coding

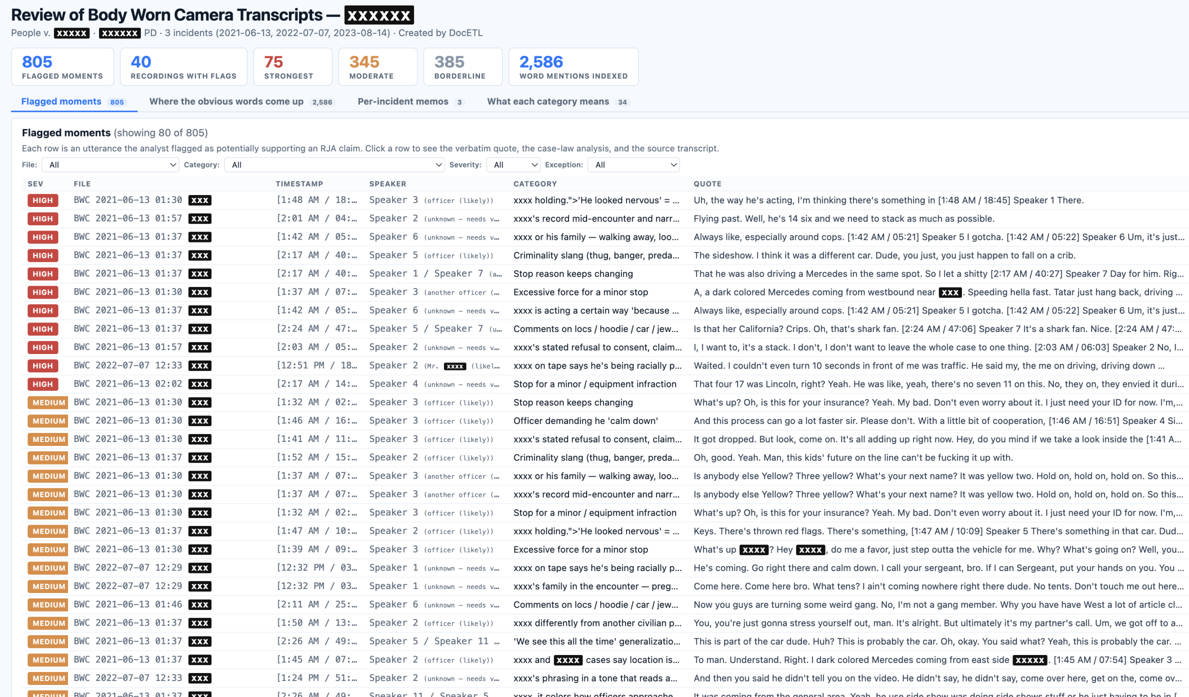

Figure 4: A custom dashboard generated for a deployment with public defenders reviewing body-worn camera transcripts for evidence of racial bias. Each row is an utterance the LLM extracted as potentially supporting a legal claim, shown with its severity, speaker, timestamp, claim category, and the quoted passage from the transcript. The layout is tailored to the query's operators and output schema.

agent and costing roughly $5 per dashboard. A more scalable direction would be to build reusable UI components for common patterns (e.g., lists for extractions, threaded views for conversations, collapsible layouts for large output schemas) and have the data system generate and compose them, with users able to provide feedback directly in the dashboard (e.g., correcting a label or flagging a bad output) that propagates back to the data system as revised prompts or constraints on future query plans.

Building systems for bespoke operators. The bespokeness of the operators users actually write has implications for how systems plan queries and are evaluated.

Even when bespoke operators resemble standard NLP tasks like extraction, classification, or summarization, they operate on custom data with domain-specific definitions, and different models turn out to be better or worse at different task types. Work from the ML community demonstrates that no single LLMs achieves the best accuracy on all possible tasks [34], and our prior work also corroborates that different models are Pareto-optimal for different types of data processing tasks [16]: for example, GPT-5 Nano is commonly selected for large-scale extraction, while more expensive models tend to be selected for summarization and report generation. Today, query optimizers discover which LLMs works best for which operator by trying multiple LLMs on a sample and comparing results, which is slow and expensive. Some prior work allows encoding priors about LLM acuracy [15], but how to develop such priors for LLM-powered data processing tasks remains unclear.

A deeper question is how pretraining and post-training impact LLM performance across different LLM-powered query processing strategies, such as how much of the document to include in the context window, whether to decompose an operator into sub-tasks, or how much prompting is needed. Prior work from the ML community has found that an LLM's accuracy on a question correlates with how

often relevant documents appeared in its pretraining data [35]. If a query optimizer knew what a model was trained on, it could make cheaper and better decisions: e.g., if the model encountered the documents during training, it may not need the full document in the context window, since the relevant content is already encoded in its parameters; if the model is already familiar with the task (e.g., sentiment classification), minimal prompting may suffice, and a smaller fine-tuned model may be equally accurate at a fraction of the cost. We do not yet have a principled way to detect what an LLM already knows about a given document or task.

Ultimately, all of the directions above share a common theme: the systems we build should be designed around how people actually use them. Users do not write a perfect query and run it once. They explore, revise, validate, and converge over many iterations, on tasks that are specific to their data and their domain. The closer our systems and benchmarks match that reality, the more useful they will be.

6 Conclusion

LLM-powered data systems have made it possible to query unstructured data at scale, but the people using them face challenges that our community has largely overlooked. Through building and deploying DocETL and DocWrangler, we found that users struggle to specify what they want, cannot easily tell whether to trust the outputs, and work around both problems through extensive iteration and creative use of LLM-powered operators. The queries they write are overwhelmingly bespoke, and our benchmarks do not reflect them. Designing systems around how people actually work, rather than how we assume they do, is the central challenge ahead. We hope the observations and directions in this article inspire our community build LLM-powered data systems that are not just powerful and scalable, but also accessible to the people who need them.

References

[1] Snowflake, "Snowflake cortex ai: Llm functions." `https://docs.snowflake.com/en/user-guide/snowflake-cortex/llm-functions`, 2024. Accessed 2025.

[2] P. Liskowski, B. Han, P. Aggarwal, B. Chen, B. Jiang, N. Jindal, Z. Li, A. Lin, K. Schmaus, J. Tayade, W. Zhao, A. Datta, N. Wiegand, and D. Tsirogiannis, "Cortex aisql: A production sql engine for unstructured data." `https://arxiv.org/abs/2511.07663`, 2025.

[3] Google Cloud, "Bigquery ml: Generative ai functions." `https://cloud.google.com/bigquery/docs/generative-ai-overview`, 2024. Accessed 2025.

[4] Google Cloud, "Alloydb ai: Work with generative ai." `https://cloud.google.com/alloydb/docs/ai/work-with-generative-ai`, 2024. Accessed 2025.

[5] K. Huang, Y. Shi, D. Ding, Y. Li, Y. Fei, L. V. S. Lakshmanan, and X. Xiao, "Thriftllm: On cost-effective selection of large language models for classification queries," *arXiv preprint arXiv:2501.04901*, 2025.

[6] S. Jo and I. Trummer, "Sparellm: Automatically selecting task-specific minimum-cost large language models under equivalence constraint," *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, 2025.

[7] L. Patel, S. Jha, M. Pan, H. Gupta, P. Asawa, C. Guestrin, and M. Zaharia, "Semantic operators and their optimization: Enabling llm-based data processing with accuracy guarantees in lotus," *Proceedings of the VLDB Endowment*, vol. 18, no. 11, pp. 4171–4184, 2025.

[8] M. Urban and C. Binnig, "Efficient learned query execution over text and tables," *arXiv preprint arXiv:2410.22522*, 2024.

[9] A. Mhedhbi *et al.*, “Beyond quacking: Deep integration of language models and rag into duckdb,” *Proceedings of the VLDB Endowment*, vol. 18, pp. 5415–5428, 2025.

[10] G. Sanmartino, M. Urban, P. Papotti, and C. Binnig, “The stretto execution engine for llm-augmented data systems,” *arXiv preprint arXiv:2602.04430*, 2026.

[11] S. Liu, A. Biswal, A. Kamsetty, A. Cheng, L. G. Schroeder, L. Patel, S. Cao, X. Mo, I. Stoica, J. E. Gonzalez, and M. Zaharia, “Optimizing llm queries in relational data analytics workloads,” *arXiv preprint arXiv:2403.05821*, 2024.

[12] S. Shankar, T. Chambers, T. Shah, A. G. Parameswaran, and E. Wu, “Docetl: Agentic query rewriting and evaluation for complex document processing,” *Proceedings of the VLDB Endowment*, vol. 18, no. 9, pp. 3035–3048, 2025.

[13] G. Xiao, E. Zhang, N. Sullivan, W. Hansen, and M. Balazinska, “Kathdb: Explainable multimodal database management system with human-ai collaboration,” *arXiv preprint arXiv:2512.11067*, 2025.

[14] C. Liu, M. Russo, M. Cafarella, L. Cao, P. B. Chen, Z. Chen, M. Franklin, T. Kraska, S. Madden, R. Shahout, *et al.*, “Palimpzest: Optimizing ai-powered analytics with declarative query processing,” in *Proceedings of the Conference on Innovative Database Research (CIDR)*, p. 2, 2025.

[15] M. Russo, C. Liu, S. Sudhir, G. Vitagliano, M. Cafarella, T. Kraska, and S. Madden, “Abacus: A cost-based optimizer for semantic operator systems,” *PVLDB*, vol. 19, no. 5, 2026.

[16] L. L. Wei, S. Shankar, S. Zeighami, Y. Chung, F. Ozcan, and A. G. Parameswaran, “Multi-objective agentic rewrites for unstructured data processing,” *To Appear at VLDB*, 2026.

[17] A. G. Parameswaran, S. Shankar, P. Asawa, N. Jain, and Y. Wang, “Revisiting prompt engineering via declarative crowdsourcing,” in *Conference on Innovative Data Systems Research (CIDR)*, 2024.

[18] S. Zeighami, Y. Lin, S. Shankar, and A. Parameswaran, “Llm-powered proactive data systems,” *Data Engineering*, p. 90.

[19] S. Shankar, S. Zeighami, and A. Parameswaran, “Task cascades for efficient unstructured data processing,” *Proc. ACM Manag. Data*, vol. 4, Apr. 2026.

[20] S. Zeighami, S. Shankar, and A. Parameswaran, “Cut costs, not accuracy: Llm-powered data processing with guarantees,” *Proc. ACM Manag. Data*, vol. 3, Dec. 2025.

[21] S. Zeighami, S. Shankar, and A. G. Parameswaran, “Featurized-decomposition join: Low-cost semantic joins with guarantees,” *To Appear at VLDB*, 2026.

[22] Y. Sun, S. Zeighami, B. Chopra, S. Shankar, and A. G. Parameswaran, “Semantic data processing with holistic data understanding,” *arXiv preprint arXiv:2604.02655*, 2026.

[23] California State Legislature, “Assembly bill no. 2542: California racial justice act of 2020.” Cal. Penal Code § 745, 2020. Approved by the Governor, September 30, 2020.

[24] Berkeley Institute for Data Science, UC Berkeley Investigative Reporting Program, and Stanford Big Local News, “California police records access project.” `https://bids.berkeley.edu/california-police-records-access-project`, 2025. Part of the California Law Enforcement Accountability Network (CLEAN) initiative.

[25] S. Shankar, B. Chopra, M. Hasan, S. Lee, B. Hartmann, J. M. Hellerstein, A. G. Parameswaran, and E. Wu, “Steering semantic data processing with docwrangler,” in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, 2025. Best Paper Honorable Mention.

[26] CalMatters and The Garrison Project, “California's racial justice act boasts few successes, four years in.” `https://calmatters.org/justice/2024/11/california-racial-justice-act/`, 2024. Reports that the state keeps no systematic count of § 745 motions; roughly a dozen successful motions statewide in the law's first four years.

[27] H. H. Clark and S. E. Brennan, “Grounding in communication,” in *Perspectives on Socially Shared Cognition* (L. B. Resnick, J. M. Levine, and S. D. Teasley, eds.), pp. 127–149, American Psychological Association, 1991.

[28] A. T. Kalai and S. S. Vempala, "Calibrated language models must hallucinate," in *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pp. 160–171, 2024.

[29] P. Sui, E. Duede, S. Wu, and R. J. So, "Confabulation: The surprising value of large language model hallucinations," *arXiv preprint arXiv:2406.04175*, 2024.

[30] Y. Liu, H. Zhou, Z. Guo, E. Shareghi, I. Vulić, A. Korhonen, and N. Collier, "Aligning with human judgement: The role of pairwise preference in large language model evaluators," in *First Conference on Language Modeling*, 2024.

[31] S. Shankar, J. Zamfirescu-Pereira, B. Hartmann, A. Parameswaran, and I. Arawjo, "Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences," in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–14, 2024.

[32] J. Lao, A. Zimmerer, O. Ovcharenko, T. Cong, M. Russo, G. Vitagliano, M. Cochez, F. Özcan, G. Gupta, T. Hottelier, H. V. Jagadish, K. Kissel, S. Schelter, A. Kipf, and I. Trummer, "Sembench: A benchmark for semantic query processing engines," *arXiv preprint arXiv:2511.01716*, 2025.

[33] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 3363–3372, 2011.

[34] T. Shnitzer, A. Ou, M. Silva, K. Soule, Y. Sun, J. Solomon, N. Thompson, and M. Yurochkin, "Large language model routing with benchmark datasets," *arXiv preprint arXiv:2309.15789*, 2023.

[35] N. Kandpal, H. Deng, A. Roberts, E. Wallace, and C. Raffel, "Large language models struggle to learn long-tail knowledge," *Proceedings of the 40th International Conference on Machine Learning*, 2023.