Steering Semantic Data Processing With DocWrangler

Shreya Shankar[1†], Bhavya Chopra[1†], Mawil Hasan[1], Stephen Lee[1],
Björn Hartmann[1], Joseph M. Hellerstein[1], Aditya G. Parameswaran[1], Eugene Wu[2]
[1]UC Berkeley EECS, [2]Columbia University
{shreyashankar,bhavyachopra,mawil0721,stephen_lee129,hellerstein,adityagp} @berkeley.edu,
bjoern@eecs.berkeley.edu, ewu@cs.columbia.edu

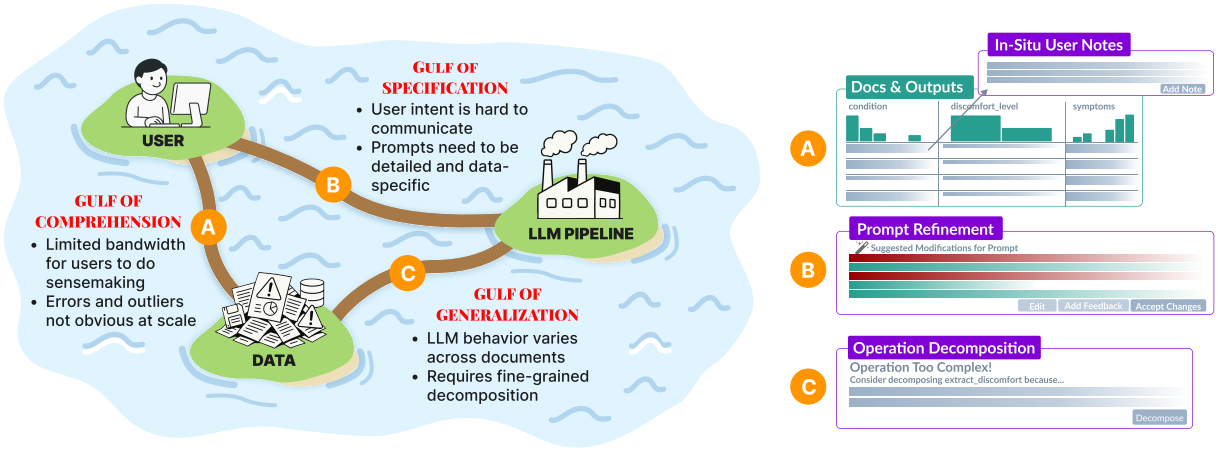

Figure 1: Effective semantic data processing requires interaction between users, their pipeline of LLM calls, and their data (both raw documents and LLM outputs). DocWrangler contributes three novel features to address: (A) the gulf of comprehension through *in-situ user notes*, (B) the gulf of specification through *LLM-assisted prompt refinement*, and (C) the gulf of generalization through *LLM-assisted operation decomposition*.

Abstract

Unstructured text has long been difficult to automatically analyze at scale. Large language models (LLMs) now offer a way forward by enabling *semantic data processing*, where familiar data processing operators (e.g., `map`, `reduce`, `filter`) are powered by LLMs instead of code. However, building effective semantic data processing pipelines presents a departure from traditional data pipelines: users need to understand their data to write effective pipelines, yet they need to construct pipelines to extract the data necessary for that understanding—all while navigating LLM idiosyncrasies and inconsistencies. We present DocWrangler, a mixed-initiative integrated development environment (IDE) for semantic data processing with three novel features to address the gaps between the user, their data, and their pipeline: *(i) In-Situ User Notes* that allows users to inspect, annotate, and track observations across documents and LLM outputs, *(ii) LLM-Assisted Prompt Refinement* that transforms user notes into improved operations, and *(iii) LLM-Assisted Operation Decomposition* that identifies when operations or documents are too complex for the LLM to correctly process and suggests decompositions. Our evaluation combines a think-aloud study with 10 participants and a public-facing deployment (available at docetl.org/playground) with 1,500+ recorded sessions, revealing how users develop systematic strategies for their semantic data processing tasks; e.g., transforming open-ended operations into classifiers for easier validation and intentionally using vague prompts to learn more about their data or LLM capabilities.

CCS Concepts

• **Information systems** → **Data management systems**; • **Human-centered computing** → **Interactive systems and tools**; • **Computing methodologies** → **Artificial intelligence**.

Keywords

Data Processing, Large Language Models, Human-AI Interaction

1 Introduction

Unstructured documents—such as PDFs, reports, transcripts, and emails—are notoriously difficult to analyze automatically [18]. Traditional data analysis systems like relational databases [29] and map-reduce [17] typically operate on structured data (i.e., tables) and fail to effectively handle unstructured content. But large language models (LLMs) now present a way forward through *semantic data processing* [4, 47, 64, 71, 91]: a paradigm where users can instruct LLMs to manipulate data through familiar data processing operators like `map`, `reduce`, `filter`, and `groupby`. For example, consider analyzing medication side effects in unstructured doctor-patient conversation transcripts. For this task, a semantic `map` could extract mentions of medications and reported side effects, followed by a semantic `reduce` to summarize effects per medication. The impact of semantic data processing could be transformative: just as traditional

†Co-first authors. Corresponding author: Shreya Shankar.

data analysis systems enabled structured data processing at scale and are now ubiquitous across sectors, semantic data processing has the potential to revolutionize how we work with data, but for the even-bigger realm of unstructured documents [21, 53].

However, current semantic data processing systems remain far from realizing their potential. As Fig. 1 illustrates, e ective semantic data processing involves three components the user, their pipeline of LLM operations, and the data (both raw documents and LLM outputs) with signi cant challenges emerging from their interactions. Drawing on Norman's gulfs of execution and evaluation [62], we identify gulfs that pertain to each pairwise interaction. First, consider the *gulf of comprehension* between users and their unstructured data. Documents contain too much information for humans to fully process [35, 42], and LLMs, when asked to process many documents, inevitably and unpredictably introduce mistakes or misinterpretations that users nd di cult to keep track of. Second, there is a *gulf of speci cation* between users and their semantic data processing pipelines: users must rst discover their true intent often only possible after exploring su cient data to understand what questions the data can reasonably answer and then express this intent as pipelines of operations with corresponding LLM prompts [33, 86]. In both cases, users struggle with distilling their many observations about data patterns and LLM behaviors into e ective speci cations. Third, the *gulf of generalization* between the semantic data processing pipeline and data persists even when a pipeline perfectly captures user intent. Even with clear, unambiguous prompts, LLMs may fail to generalize correctly to the user's actual data, struggling with long documents [49] or complex operations requiring simultaneous reasoning about multiple elements [33]. Complex tasks need to be decomposed into multiple LLM calls instead of one [20, 71, 79].

To address the aforementioned gulfs, we present DocWrangler , a mixed-initiative integrated development environment (IDE) designed for semantic data processing (Fig. 2). DocWrangler builds on established interaction patterns for transforming data: a notebook-like pipeline constructor with operation cells that can be reordered and toggled [11], and a spreadsheet-like output viewer with automatically-generated visualizations based on data types [35]. We additionally contribute three novel features that each address a gulf in Fig. 1. Our *in-situ user notes* feature tackles the comprehension gulf by enabling users to annotate observations directly on both documents and outputs, creating a persistent, searchable record that helps track patterns across complex datasets with minimal context switching. Our *LLM-assisted prompt re nement* feature addresses the speci cation gulf through an interactive interface where an LLM analyzes the pipeline, documents, outputs, and user notes to suggest more e ective prompts that the user can then tweak if they would like. Finally, our *LLM-assisted operation decomposition* feature targets the generalization gulf by identifying when the pipeline is inadequate for the documents, using an LLM-as-judge that runs in the background [72, 105]. Users receive noti cations highlighting detected issues along with actionable suggestions for breaking operations into more manageable steps.

To understand how DocWrangler supports semantic data processing in practice, we evaluated its use as a design probe, focusing on user needs, strategies, and work ows. Our evaluation used two complementary approaches: (i) a think-aloud study with 10 participants across di erent domains that provided rich qualitative insights into users' mental models and decision-making processes, and (ii) a public online deployment (available at docetl.org/playground) that attracted over 1,500 uses, allowing us to quantitatively analyze pipeline structures and how they changed. We observed that users develop systematic strategies for their analysis tasks, for example, by transforming open-ended operations into classi ers for easier validation. Users also employed intentionally vague prompts in `map` operations to learn more about their data, reminiscent of epistemic actions, i.e., actions taken not to directly achieve a goal but to gather information that reveals new possibilities [40, 89].

In summary, we contribute the following:

- The design and implementation of DocWrangler , an IDE for semantic data processing;
- Findings from a 10-person think-aloud study that demonstrate how users opportunistically align with LLMs, with rich insights into both progress and challenges; and
- Insights from a real-world deployment showing how semantic data processing is used across domains and areas where users need additional support.

We rst review related work in Section 2. We next give a more detailed background on operators in semantic data processing systems and motivate DocWrangler 's design in Section 3. Subsequently, we describe our interface and implementation in Section 4. Then, we present our study methodology in Section 5. We detail our ndings from our user study in Section 6 and deployment in Section 7, and discuss implications for future system design and human-AI collaboration in Section 8.

2 Related Work

In this section, we review relevant work that informs DocWrangler 's development: semantic data processing systems that enable LLM-powered operations on text, then, interfaces for structured data analysis, and nally, solving general tasks with LLM pipelines.

Semantic Data Processing Systems. The goal of semantic data processing is to leverage LLMs to process large volumes of unstructured text through natural language instructions. For instance, a pipeline might use an LLM-powered `map` operation to classify sentiment in customer reviews, followed by an LLM-powered `reduce` operation that summarizes common themes for each class (i.e., positive and negative). While several systems now support semantic data processing speci ed through code [4, 47, 64, 71], users struggle to build e ective pipelines with these systems [4, 48]. Some systems try to automatically optimize pipelines for cost [47] or accuracy [71]. However, the end-user challenges of determining what data processing intent to express and how best to express it remain unsolved, given that user preferences may depend on LLM outputs themselves [72]. Moreover, automatic accuracy optimization techniques often require extensive time and computational resources to explore di erent decompositions and prompts [38, 71, 78], making them impractical for interactive settings. Generally, semantic data processing systems focus on optimizing an *inner* computational loop that assumes well-speci ed tasks and clear evaluation metrics. However, real-world semantic data processing rarely begins with perfectly formed speci cations, and current systems provide

Figure 2: Screenshot of the DocWrangler IDE showing our notebook-style pipeline constructor (A) and spreadsheet-like data inspector (B). User notes on documents and LLM-generated data, provided in-situ, and the raw document collection are displayed in the left and right sidebars, respectively.

minimal support for the crucial human-centered outer loop of discovery and refinement. DocWrangler provides such an environment for users to author and test their semantic data processing pipelines through iterative exploration.

Interfaces for Structured Data Analysis. The history of data analysis tools provides valuable insights for building semantic data processing interfaces. We briefly cover interaction paradigms for data analysis, all of which operate instead on structured data, as semantic data processing of unstructured text is a new field.

First, coding libraries, from early ones like SAS [45] to modern popular ones like Pandas [54], provide data transformation capabilities through API calls. While such libraries provide high flexibility, they require users to constantly switch between programming environments and data inspection tools. Graphical data flow interfaces [32] improved upon this workflow by visualizing processing steps through boxes-and-arrows metaphors, though they still obscure the actual data being transformed. Spreadsheets [80] and visual analytics tools [31] instead opt for direct manipulation for editing and visualizing data respectively, making data analysis more accessible to non-programmers. Over the decades, research has consistently shown that always-on visualizations can complement data processing tools, helping users discover insights and validate transformations faster [34, 44, 75], especially when users can directly interact with visualizations [76].

Hybrid approaches aim to combine the best aspects of the previous paradigms. Systems like Potter's Wheel [65] combine interactive interfaces with underlying domain-specific languages for data transformation. Programming-by-example approaches [25, 99] infer transformations from user demonstrations in limited settings (e.g., regex-based transformations), though for enterprise data processing they require many examples to get started, and are difficult to validate and generalize. Predictive interaction [28] introduced a mixed-initiative approach to data transformation through a guide/decide loop, where systems present suggested transformations based on user selections, while also letting users inspect data to see which transforms are necessary [35].

More recently, LLMs have transformed how users interact with data, though we are still in the early stages of this evolution. The first generation of LLM-based tools primarily created natural language interfaces for structured data tasks: specifically, conversational assistants that generate SQL or Pandas code [24, 26, 69, 104]. Steering these AI-generated pipelines is difficult [55]: LLMs often misinterpret user intent or produce incorrect code, requiring specialized interfaces for correction. Xie et al. [98] propose visualizing pipelines as directed acyclic graphs, reminiscent of data flow GUIs, while Kazemitabaar et al. [37] propose a notebook-style interface that breaks complex workflows into discrete, inspectable steps. These environments are code-centric: they prioritize code review over data inspection, overlooking the well-established value of integrated data visualization [44, 76, 97]. Since data is messy and transformations rarely work on the first try, users need to switch between writing code and reviewing outputs to catch anomalies and spot data that still needs processing.

Moreover, perhaps because they focus on structured data, prior work treats LLMs merely as code generators rather than semantic operators in their own right. In semantic data processing, LLMs aren't just writing scripts in a traditional data processing language, they provide entirely new black-box capabilities for unstructured data transformations. This creates a new challenge: while traditional data processing code can often be validated through static

analysis or test cases, LLM behavior in semantic operations is inherently uncertain and context-dependent. Effective interfaces for semantic data processing must therefore enable refinement of both the pipeline structure and individual operations while keeping data continuously visible throughout the process.

LLM Workflow Development and Validation. As LLMs tackle increasingly complex problems, like semantic data processing, users need multi-step workflows to express their goals. Effective interfaces for general problem-solving with LLMs should support the entire process from task decomposition to output refinement building on established research in mixed-initiative interfaces [31], collaborative AI [2], and interactive machine learning [19].

Task decomposition has deep roots in both human and AI systems. Crowdsourcing research introduced strategies for breaking down complex tasks, some of which has been successfully applied to LLMs [23, 63], and cognitive science has studied human approaches to problem decomposition [59]. Recent interfaces apply these insights to convert natural language into executable subtasks [37, 103], but users still struggle with information overload when examining LLM-generated workflows [12, 37, 100]. This points to a critical need for interfaces that help users make sense of these complex workflows. Some tools address pipeline sensemaking by enabling interactive prompt experimentation and output inspection [5, 96], although these are not designed for data processing. Unlike general LLM pipelines, semantic data processing faces steeper challenges in bridging the gulfs shown in Fig. 1. The heterogeneous nature and scale of unstructured documents make both comprehension and LLM generalization more difficult. Separately, while semantic data processing is underexplored, we are starting to see point solutions address specific semantic data processing applications; e.g., LLooM's interface for concept induction tasks [42]. However, we lack general-purpose interfaces for semantic data processing across diverse document and operator types. Designing such interfaces is not straightforward, as users encounter the gulf of envisioning [83]: the cognitive gap between having a goal and translating it into effective LLM instructions while also understanding how to evaluate whether the output meets their original intentions.[1]

Even when a complex task is expressed as a pipeline of well-scoped operations, validating and debugging LLM outputs remains challenging [5, 16, 50]. Users face high cognitive load from constantly switching between prompting and evaluation [82, 87], and developers struggle with LLMs' unpredictability: fixing one issue often creates others [101]. Some tools address these challenges through automated evaluation, often employing an LLM-as-judge methodology where an LLM evaluates outputs against specific criteria [39, 51, 70, 72, 85, 105]. These automated evaluation approaches, which focus on validating individual outputs, have not been considered in the context of semantic data processing, where users need to assess both individual results and aggregate patterns across entire datasets. Moreover, in semantic data processing, different types of operations require distinct evaluation approaches: for example, assessing the accuracy of information extraction is different from assessing the quality of a summary. Specialized visualizations [22, 84] can help users validate and understand LLM behavior, though again, these have not been applied to semantic data processing. Additionally, identifying errors in LLM outputs is only half the battle. Users also need tools to refine pipelines based on observed patterns. We build on prior work that proposes automatic prompt refinement for image generation [1, 10]: for semantic data processing, prompts may need to change in ways beyond specific attributes (e.g., subject or style) as users discover their true question, potentially shifting the entire task direction.

Summary. Overall, prior work points to a clear challenge: semantic data processing systems lack effective interfaces for users to iteratively develop, validate, and refine their pipelines. Prior work also sheds light into what we should care about when building an interface for semantic data processing workflows: treating them as data transformation workflows first and foremost, while simultaneously providing tools for effective prompt engineering, enabling users to validate both individual outputs and aggregate patterns across documents, and helping users navigate the gulfs proposed in Fig. 1. DocWrangler embodies the aforementioned principles as an IDE, while preserving users' agency throughout [3, 31].

3 DocWrangler Design

We now describe the foundation and design principles of DocWrangler. While our interface builds on DocETL [71], an open-source semantic data processing system, as the backend, we could have equally well chosen to build DocWrangler on top of other semantic data processing systems [4, 47, 64, 91]. We first provide background on DocETL, then present our design goals based on an analysis of user needs and challenges in semantic data processing.

3.1 DocETL Background and Example

DocETL is a declarative framework for building semantic data processing pipelines, where many of the unit data processing operations are executed by LLMs. Each LLM-powered operator is defined through two components: a natural language *prompt* that specifies what the operation should do, and an *output schema* that determines the structure of data the LLM should generate. Inputs to DocETL are documents, represented as JSON collections of key-value pairs, allowing flexible handling of both structured and unstructured content. An entirely unstructured document (like an email or news article) can be represented as a single key-value pair (e.g., `{"content": "full text here..."}`). DocETL offers multiple semantic operators, including `map` (applies prompts to individual documents), `filter` (removes documents that don't meet specified criteria), `reduce` (processes document groups collectively), and `resolve` (performs entity resolution and canonicalization across documents). These operations leverage Jinja templates in their prompts [68], enabling users to reference specific document key-value pairs. As the pipeline executes, each operation enriches documents with new key-value pairs according to its output schema, allowing subsequent operations to build upon earlier results.

To illustrate how these components work together in practice, consider a semantic processing pipeline analyzing student course reviews to identify common themes of complaints, with supporting evidence. The pipeline might consist of three operations:

[1] Note that for semantic data processing, Subramonyam et al.'s [83] gulf of envisioning encompasses all the gulfs presented in our model in Fig. 1.

(1) A `map` operation that processes each review (represented as a separate JSON document) individually, with a prompt asking the LLM to `Extract complaint themes and their supporting quotes from this review` and an output schema defining two new attributes, `themes` (an array of strings) and `supporting_quotes` (an array of strings corresponding to each theme).
(2) A `resolve` operation that semantically de-duplicates themes across all reviews (e.g., recognizing that `professor talks too fast` and `professor speaks quickly` represent the same underlying complaint), with a comparison prompt to `Consider if these two themes are similar` and return `True` if so; and a resolution prompt to `Output a single theme that best represents these themes`.
(3) A `reduce` operation that groups reviews by theme and generates a summary report for each theme, with a prompt like `Summarize the common sentiments and representative quotes for this theme`, and an output schema for the new `summary` attribute.

Although a system like DocETL can relieve users of the low-level execution details (e.g., orchestrating all LLM calls associated with such a pipeline), constructing effective pipelines remains challenging because users often cannot fully specify their analytical intents in advance. For example, what the LLM believes is a complaint may not be the type or granularity of complaint the user is interested in, or the themes extracted might combine issues the user would prefer to see separately.

3.2 DocWrangler Design Goals

To better understand the challenges users face when building LLM-powered data processing pipelines, we analyzed messages and feedback from the DocETL Github and Discord community[2] (400+ members) and drew insights from prior work (described in Section 2).

We observed that users' workflows in semantic data processing typically follow three phases that we call the Three I's. Users cycle between *initializing* their pipelines by defining operations, then *inspecting* outputs to understand results, and *improving* their pipelines based on these insights. We organize user pain points across these phases. First, there are *initialization* challenges. Writing effective operations requires predicting how well an LLM will interpret a user's data processing intent. This challenge creates a premature commitment problem [9]: users invest time crafting operations without knowing if they'll work, often wasting effort when outputs fail to meet expectations. Users expressed a desire for assistance in writing the pipeline, as well as preview functionality to validate their approach before committing to full execution.

Next, there are *inspection* challenges. After initializing, users struggle to validate whether LLM behavior across documents aligns with their expectations for each operation. The volume of data makes review of all documents and LLM outputs impractical. Users often don't know when they're "losing the forest for the trees" or vice versa when reviewing outputs. Users expressed wanting to review intermediate operation outputs, and some mentioned that they exported outputs as spreadsheets to look at results as different views (row by row, all at once in a spreadsheet, or side by side) in an ad-hoc manner. Fragmenting the validation workflow across multiple tools creates additional cognitive load for users [15, 36].

[2] https://discord.gg/fHp7B2X3xx

Third, there are *improvement* challenges. Users struggle to externalize patterns both positive and negative observed across multiple documents. Some users took notes in spreadsheets or a separate text file as a way to organize their insights, but they struggled to translate their notes into concrete pipeline modifications. Prior work also observes that users struggle to prompt LLMs to edit pipelines [37]. Moreover, users did not know how to improve pipelines when the task was "too hard" for the LLM, requiring operation decomposition, or a redistribution of tasks between LLM-powered and code-powered operators.

We also turned to prior work in structured data processing interfaces to understand what users might need in an semantic data processing interface. Visualization research demonstrates that users need both overview and detail views to efficiently make sense of complex datasets [73], i.e., showing aggregates first helps users identify patterns, while enabling drill-down into specific examples supports verification [43, 81]. Data wrangling tools demonstrate improved user productivity when users can see and edit data transformation code directly [35]. Mixed-initiative interfaces highlight the importance of preserving user agency, ensuring system observability, and reducing context-switching costs when supporting complex, iterative tasks [3, 31, 74].

Overall, we formulated five design goals for an effective semantic data processing interface:

D1. Scaffold Pipeline Initialization: Help users create and configure operations with minimal friction, with built-in guidance and quick experimentation.
D2. Facilitate Efficient Data Inspection and Notetaking: Enable users to validate inputs and outputs individually and in aggregate, while supporting note-taking to capture insights and patterns.
D3. Guide Pipeline Improvement: Offer assistance for translating user feedback into effective pipeline modifications, both at the individual operation level (e.g., prompt improvements) and pipeline level (e.g., operation decomposition).
D4. Maintain End-to-End Observability: Ensure transparency into transformation logic at each pipeline step (e.g., inputs, outputs, LLM prompts).
D5. Minimize Context Switching: Integrate all essential analytical capabilities within a unified interface, minimizing the need for external tools (e.g., spreadsheets, custom scripts, AI assistants like ChatGPT).

4 DocWrangler System

We present DocWrangler, an integrated development environment (IDE) for semantic data processing. First, we provide an overview of the solution and how it addresses our design goals. Then, we present an example usage scenario that walks through our features. Finally, we describe technical implementation details.

4.1 Overview of Solution

DocWrangler's interface (Fig. 2) integrates a file and notes viewer, pipeline editor, and dataset viewer in a unified workspace. To scaffold pipeline initialization (D1), we developed a notebook-inspired [41] editor with interactive operation cards featuring syntax validation, drag-and-drop reordering, and the ability to toggle

Figure 3: Work ow for analyzing patient discomfort from medical transcripts (D1; D2). (A) The user adds a new operation via dropdown menu. (B) A map operation card appears with syntax validation. (C) The user writes a prompt for extracting discomfort information. (D) The user de nes an output schema with three attributes to be extracted: discomfort level, description, and symptoms. (E) Before running the full dataset, they sample 10 documents. (F) Results appear in an interactive table. (G) Eye icons reveal exact LLM prompts for each document. (H) Column headers visualize attribute distributions. (I) Sort buttons reorder documents. (J) Adjustable column widths help focus on speci c content. (K) Search functionality helps validate extractions.

Phase	Feature	Goals
Initialize	Pipeline editor with operation cards that can be toggled and reordered	D1, D4
	Custom cards for each operator type with live syntax validation	D1
	Visual document ow between operations to show transformation paths	D1, D4
	Execution on sampled documents for quick iteration	D1
Inspect	Spreadsheet-style viewer with automatic visualizations per column	D2
	Support for column resizing, ltering, sorting, and search	D2, D5
	Per-document prompt viewer to inspect exact LLM inputs	D4
	Detailed document viewer with keyboard navigation and side-by-side comparison	D2, D5
	Output inspection available at any pipeline stage, including intermediates	D2, D4
Improve	In-situ note-taking system for annotating and categorizing output issues	D3, D5
	Persistent notes sidebar with ltering and search across iterations	D3, D4
	AI-assisted Prompt Re nement interface driven by user notes	D3, D5
	Automatic operation decomposition with explanations and plan di s	D3, D4
	Embedded AI assistant for help with prompts, templates and work ow guidance	D1, D3, D5

Table 1: Key features in DocWrangler , grouped by phase of semantic data processing, and linked to design goals.

operations on and o (Fig. 2A). Once initialized, we allow users to run pipelines on samples to reduce response time, and we cache all intermediate results for future reuse. After execution, it is to inspect outputs e ciently (D2), our spreadsheet-like viewer, as shown in Fig. 2B provides automated visualizations of document attributes as columns, along with sorting, ltering, resizing, and search capabilities. Our design is reminiscent of data wrangling interfaces that also show high-level summaries [35]; however, most LLM-generated attributes are unstructured text, so we show histograms of word or character counts as appropriate. While not depicted in Fig. 2, DocWrangler also has a detailed document inspector that allows users to examine LLM outputs and source documents row by row, with side-by-side comparison, as will be described in Section 4.2.

Then, to guide pipeline improvement (D3), we introduce three novel features: (i) In-Situ User Notes enables users to capture observations directly during inspection of documents and LLM outputs, with notes persisting across pipeline runs; (ii) Prompt Re ne ment suggests improved versions of prompts, based on in-situ notes, through a conversational interface that supports both direct editing and AI-assisted revisions; and (iii) Operation Decomposition proactively identi es when operations exceed LLM capabilities and o ers automatic restructuring with explanations. Additionally, to maintain end-to-end observability (D4), we visualize document ow between operations, provide access to exact LLM prompts, and make intermediate outputs and suggested Operation Decomposition feedback inspectable. Finally, to minimize context switching (D5), we integrated all analytical capabilities within a single interface and, following recent systems [37, 103], included a context-aware AI assistant to help users navigate the

Figure 4: In-situ user notes feature in DocWrangler (D2). (A) User selects attribute to inspect. (B) Document viewer dialog shows attribute statistics and enables in-situ notes. (C) User could inspect documents side-by-side with split-screen view, if they want. (D) User adds a note. (E) Notes persist in the main interface sidebar.

system. Table 1 summarizes the key features of DocWrangler across each of the three I's and how they map to our design goals.

4.2 Feature Walk-Through

We present our features by illustrating an example DocWrangler scenario, where, given a corpus of doctor-patient conversation transcripts, a medical data analyst wants to analyze how emotionally comfortable and open patients are during their visits, and how this varies by symptom reported.

4.2.1 Initialization Phase (D1). The analyst begins by uploading the collection of doctor-patient transcripts through the upload interface (top left of Fig. 2). They can inspect their raw data in the dataset viewer on the right, which provides an overview through statistics about the transcript collection, including document count (here, 87), word count distributions, and existing metadata fields (D4, D5). With this initial understanding of their data, the analyst creates their first operation by clicking the Add Operation button (Fig. 3A), selecting a map type operation, which will create new attribute(s) for each document. While defining the operation (Fig. 3B, Fig. 3C), they write a prompt instructing the LLM to Extract discomfort information from the medical transcript. {{ input.src }} Identify the discomfort level (low, medium, high), provide a brief description of the discomfort, and list the symptoms the patient complains about. The analyst then defines three new attributes for the LLM to populate (Fig. 3D). Before committing to processing their entire dataset, they enable document sampling (10 documents; Fig. 3E) and click Run to quickly test their pipeline.

4.2.2 Inspection Phase (D2). The analyst reviews results in the table view (Fig. 3F), where each row represents a document and columns show attributes. Hovering over eye icons for each row (Fig. 3G) reveals the exact LLM prompt, enabling verification that the Jinja template was specified correctly (D4). Noticing the unexpected distribution of discomfort levels in column headers (Fig. 3H) mostly medium or high the analyst investigates by sorting documents (Fig. 3I), resizing columns (Fig. 3J), and scanning descriptions. This reveals a misalignment: LLM outputs are focused on patients' physical symptoms rather than the analyst's intended measure of patient openness and comfort level during the visit which the analyst confirms by searching for specific terms in source documents (Fig. 3K).

4.2.3 Improvement Phase (D3). After reviewing outputs in the table, the analyst selects an attribute for closer examination (Fig. 4A), opening a full-screen dialog with document navigation capabilities (Fig. 4B). Using the In-Situ User Notes feature, the analyst adds a note in the inspector panel (Fig. 4D): the discomfort level should be about how comfortable, behaviorally, the patient is (not about the physical symptoms per se) and tags it as red.[3] After reviewing multiple documents, they return to the main interface where all notes are accessible, searchable, and iterable (Fig. 4E).

The analyst initiates the Prompt Refinement workflow by clicking Improve in the operation card (Fig. 5A), opening a dialog with their current prompt and any relevant notes (Fig. 5B). After providing any optional additional instructions and clicking Continue to Analysis, the user gets an improved prompt emphasizing behavioral aspects of discomfort (Fig. 5C), visualizing changes between versions. The analyst can directly edit the suggestion (Fig. 5D) or request further AI modifications (Fig. 5E) before saving.

4.2.4 More Iteration. After running the pipeline again (Fig. 5F), the analyst notices improved discomfort level distributions (Fig. 5G), with many documents now correctly classified as low discomfort. The analyst moves on to analyzing the extracted symptoms. While they are inspecting symptom outputs, DocWrangler notifies the user (Fig. 6A) that the operation may be too complex (D3). Clicking on the notification triggers the the Operation Decomposition feature. A dialog appears, showing examples of incorrect LLM results when handling both discomfort assessment and symptom extraction simultaneously (Fig. 6B). The analyst clicks Automatically Decompose (Fig. 6C), and the system transparently streams its accuracy optimization process (Fig. 6D), evaluating different candidate plans with LLM-as-judge evaluators[10] (D4). After a few minutes, DocWrangler then shows a visualization of the restructured pipeline (Fig. 6F) that processes the same task but divides the work across multiple operations first breaking the data into manageable chunks with a split operation, processing each chunk separately with the original map operation, and then using a reduce operation to unify the results.

[3] Users can color-code notes (e.g., red for critical issues, green for positive observations) similar to qualitative analysis tools [90, 94].

Figure 5: Prompt Refinement workflow (D3). (A) User initiates improvement via Improve button. (B) A dialog opens, showing the current operation and the user's relevant in-situ notes. (C) AI suggests improved prompt addressing notes (focusing on behavioral discomfort). (D) Improved prompt includes behavioral indicator examples. (E) User can edit the suggestion directly, or instruct the AI to edit it. (F) User runs updated operation. (G) Results show a more accurate discomfort distribution.

Figure 6: The Operation Decomposition workflow (D3). (A) A notification suggests decomposing a complex operation, simultaneous extraction of discomfort and symptoms for long documents. (B) A dialog explains why the operation exceeds LLM capabilities, with examples of inconsistent outputs. (C) User can automate decomposition or ignore. (D) System shows real-time optimization process with reasoning (e.g., evaluation criteria for determining the best decomposition). (E) The resulting pipeline splits documents into chunks, executes the original operation on each chunk, and has another operation to unify results.

4.3 Implementation Details

Here, we describe DocWrangler 's implementation details. DocWrangler is built with Next.js and TypeScript on the frontend (styled with Tailwind CSS), and a Python-based FastAPI backend. The frontend uses Monaco Editor for writing operations and ReCharts (a React wrapper around D3) for visualizations. The backend compiles the visual pipeline into DocETL's execution format and processes documents in parallel, streaming real-time updates via WebSockets. Pipeline de nitions, user notes, and datasets are stored on disk (or S3 for the public deployment). User notes are additionally cached in browser storage for faster querying. To optimize performance, we use Python's `diskcache` library for caching operation outputs, keyed by document ID and operation sequence hash. Only modi ed operations and downstream steps are recomputed. The output inspector automatically generates visualizations based on attribute type [58, 93]. For numerical attributes, we render 7-bin histograms. For boolean attributes, we use 2-bin bar charts. For string data, we rst assess whether it is categorical (fewer than 50% unique values). If so, we display bar charts of the top 7 values; if not, we show word counts for multi-word outputs and character counts for single-word outputs. All charts use virtual scrolling to support large datasets.

DocWrangler is containerized via Docker for open-source use, and hosted on Modal Labs' cloud platform for the public version. Users provide their own LLM API keys. In the following paragraph, we describe implementation details for the AI-assisted features.

Assisted Prompt Re nement. Our Prompt Re ne ment feature relies on a representation of user notes. We store user notes as a list of tuples, where a note is a tuple containing: operation identi er, attribute name, free-text comment, and an optional category tag for color-coding. Our system uses a conversational AI interface but presents a specialized revision UI rather than a traditional chat (Fig. 5E). The interface displays only the latest AI-suggested prompt, highlighting the di , while maintaining conversation context in the background. When users initiate prompt re nement, DocWrangler gathers in-situ user notes relevant to the operation and the corresponding documents, then sends an AI assistant, powered by gpt-4o-mini, a message containing the current prompt, output schema, a sample of documents, in-situ user notes and corresponding documents, and basic prompt engineering guidelines (i.e., be clear, unambiguous, and provide few-shot examples if possible). We also instruct the AI assistant to return structured content with new prompts enclosed within `<prompt>` and `</prompt>` tags and schema changes within `<schema>` and `</schema>` tags for easy parsing. The AI's response is streamed in real-time.

Whenever users edit a prompt directly or provide feedback to the AI assistant, we append a new message to the conversation history. For direct edits, we create a canonical message recording that the user changed X to Y; for feedback, we include the user's instruction to the LLM. All revisions are managed in a tree structure, visualized as an interactive diagram, where each node represents a prompt version. This diagram appears when the user clicks the Add Feedback button (Fig. 5E). Users can navigate to any previous revision point and create new branches as needed. For a more detailed screenshot of the revision tree, see Fig. 11 in Appendix A.

Table 2: Participant demographics and study tasks.

ID	Background	Dataset	Tasks
P1	Data Science, AI	Medical Transcripts	T1, T4
P2	Data Visualization	Medical Transcripts	T2, T3
P3	Operations Management	Medical Transcripts	T1, T4
P4	Data Science	Medical Transcripts	T2, T3
P5	Machine Learning Engineering	Safety Records	T1 T3
P6	Data Science	Pres. Debates	T2, T4
P7	Data Science	Medical Transcripts	T2, T3
P8	Data Science	Privacy Policies	T1, T2
P9	Medicine	Medical Transcripts	T2, T3
P10	Data Science	Pres. Debates	T1, T3

Automated Operation Decomposition Assistance. After each pipeline run, DocWrangler automatically evaluates output accuracy using an LLM-as-judge approach in the background [72, 105]. DocWrangler samples ve resulting documents and queries gpt-4o-mini in a single prompt to assess whether outputs meet criteria, returning `True` or `False`. For `False` results, DocWrangler then queries the LLM again to generate speci c failure reasons and improvement suggestions, which are displayed as non-intrusive noti cations. When users accept decomposition suggestions, DocWrangler invokes DocETL's accuracy optimizer to generate and test multiple candidate plans, or decomposed versions of the operation, and return the highest-accuracy plan (according to the LLM-as-judge). All optimization logs are streamed to the UI in real-time.

Managing Context Windows for AI-Assisted Features. Our AI-assisted features (re nement, decomposition, and the general-purpose chatbot) must handle limitations of the context window, or the maximum input size an LLM can process in one request. Since we include sample documents with every LLM interaction to provide necessary background, our messages often exceed the context window limit (128,000 tokens for gpt-4o-mini). To address this, we dynamically reduce content size before invoking the LLM. We rst calculate the total token count of the entire conversation using `gpt-tokenizer` [11]. If it exceeds the limit, we determine how many tokens to remove and distribute this reduction equally across the sample documents in the rst message only maintaining subsequent conversation history intact. For each document, we remove text from the middle while preserving beginnings and endings, replacing the removed content with an ellipsis. Essentially, as the conversation history grows, documents progressively lose more middle content to accommodate new messages within the context window. We speci cally preserve document beginnings and endings because introductions typically contain key metadata and conclusions often summarize content, both important for maintaining document context for the LLM.

5 User Study

Using DocWrangler as a design probe, we sought to understand both the tool's e ectiveness and how people make progress in semantic data processing through a task-based, think-aloud study.

Participants and Recruitment. We recruited 10 participants via a call on the DocETL Discord server. While small, this size is corroborated by prior work suggesting that even ve participants can uncover valuable usability insights [60]. All participants had prior experience using LLMs, with varied backgrounds in data processing. Four had previously used DocETL. Roles included software and ML

Table 3: Datasets and tasks used in our study.

Dataset	Tasks
Medical Transcripts	T1: Extract patient name, age, and gender T2: Analyze known risk factors for illnesses T3: Analyze symptoms and associated medical advice T4: Analyze patient discomfort vs. illness type
Presidential Debates	T1: Extract humorous quotes T2: Track how discussion of topics changes over time T3: Identify evaded questions/topics T4: Identify topics discussed by each party
Safety Records	T1: Extract incident location T2: Extract involved persons T3: Classify report behavior type
Privacy Policies	T1: Extract CCPR and GDPR mentions T2: Extract retention durations with cited regulations

engineers, data scientists, startup executives, medical professionals, and graduate students in CS and social science. All participants consented to audio and video recording. Table 2 summarizes their backgrounds and datasets.

Study Protocol. Following approval from our Institutional Review Board (IRB), we conducted one-hour long task-based sessions with each participant via video-conferencing over Zoom. Each session began with a 15-minute onboarding demo using a simple map-reduce pipeline. This walkthrough introduced DocWrangler prerequisites and features: dataset inspection, pipeline construction, pipeline execution, output inspection, and use of the AI assistant chatbot. Then, participants were given a choice between two datasets medical transcripts and prosidential debates for the study tasks. Six participants selected the former, while two participants selected the latter. Two participants brought their own document datasets for the study; P4 brought a dataset on public safety records, while P8 brought a dataset with web-scraped privacy policies. Participants were then asked to complete at least two prede ned tasks on their chosen dataset (Table 3). They spent 40 or more minutes completing tasks, and were encouraged to think aloud and ask questions as needed. After the tasks, we collected feedback through open-ended re ections and Likert-scale ratings on IDE usability and comfort. We also asked follow-up questions about challenges with LLMs and expressing intent in DocWrangler .

Analysis. We used Zoom's auto-generated transcripts, supplemented by our notes, to document each session. Four authors independently conducted open coding of notes and transcripts, followed by two rounds of axial coding [88] to identify recurring themes.

6 User Study Findings

In this section, we discuss our qualitative ndings from our study. Informally, we were interested in studying how participants design, iterate, evaluate, and debug their pipelines in DocWrangler to navigate the gulfs of comprehension, speci cation, and generalization in Fig. 1. All participants (= 10) found the IDE to be useful, and appreciated DocWrangler 's potential for diverse document analysis tasks. Participants rated the ease of using DocWrangler for the semantic data processing task highly on a 7-point Likert scale (median = 6.5, mode = 7), with 80% selecting 6 or 7 and no ratings below 5. Our key ndings are as follows:

Figure 7: Participant actions by phase: Initialize (create/edit operations), Inspect (review outputs), and Improve (re ne prompts or decompose operations). Engagement was balanced across phases, with frequent transitions between them.

- Users manipulate semantic operations to aid validation, by requesting explanatory rationales or overly speci c output attributes, and transforming open-ended tasks into structured classi cation problems;
- Users opportunistically realign their pipelines by discovering both limitations of and possibilities with LLMs, pivoting between task re nement and goal reformulation; and
- Users struggle with making sense of LLM-generated outputs at scale, requiring better provenance tracking and visualization tools tailored to semantic data processing.

We subsequently expand on our ndings.

6.1 Users Manipulate Semantic Operations to Bridge the Gulf of Comprehension

While participants di ered in how they constructed their pipelines most (= 7) built and inspected outputs of one operator at a time, while others (= 3) created complete map-reduce (i.e., extract-summarize) pipelines before inspecting outputs participants similarly repurposed semantic operations in creative ways to make sense of outputs. We observed two main approaches: modifying LLM outputs to be easier to scan and interpret, and converting open-ended tasks into more structured classi cation problems.

6.1.1 Users modify output content and forma ing to make it easier to interpret LLM behavior. To better understand LLM behavior at a glance, participants often adjusted operation outputs for interpretability. P1 and P8 added reasoning attributes to operation output schemas, with P1 noting this would force the LLM to explain its process. P6 created separate map operations to generate summaries of extracted attributes, observing that these could reduce the amount of data that you're [manually] processing and validating in the pipeline, considerably. Participants also adjusted the presentation of outputs to support manual validation. Some (P2, P3) used In-Situ User Notes to request output reformatting (e.g., bulleted lists), making results easier to scan. P8 added boolean indicator attributes (e.g., has_GDPR_mention) to lter by speci c mentions in the output table, then applied a reduce operation to summarize the behavior of the preceding map without reviewing its full output

manually. Importantly, these added attributes (e.g., rationales, summaries, indicators) were not used as final task outputs. Instead, they served to help participants verify whether the LLM had correctly interpreted their intent bridging the specification gulf illustrated in Fig. 1. Reviewing these structured outputs often triggered In-Situ User Notes, especially when outputs exposed interesting or ambiguous patterns. For instance, P9 (a doctor) noticed frequent mentions of "back pain" when operation outputs were formatted as bulleted lists. Drawing on their medical knowledge, they recognized this as a commonly reported symptom and filtered outputs containing "back pain" in the output table viewer to read mentions of "back pain" in the original documents and analyze its context with other reported conditions. They then annotated those outputs using In-Situ User Notes to clarify context for the LLM; e.g., explaining the difference between "acute back pain" and "chronic back pain"; the latter co-occurring with long-term conditions.

6.1.2 Users transform open-ended tasks to classifier-like tasks. To make validation more manageable, participants often reframed open-ended tasks as classification problems (P1, P4, P5, P8, P9, P10). P1 described this as "treating the LM like a classifier" examining class-based outputs to check for meaningful differences. For example, when analyzing doctor-patient trust, P1 initially used a free-form `trust_summary` attribute, but added a boolean `trust` attribute to validate results more easily via a histogram. As shown in Fig. 8, the LLM labeled all examples as "true," so P1 switched to a 5-point Likert scale for more granularity. While the scores were still skewed high, the distribution revealed more variation. P1 noted that they could apply a code-based rule (e.g., score >4) to interpret the results as binary, and used the differences between low and high scores to identify behavioral signals (e.g., patient stuttering), which they annotated using In-Situ User Notes.

Even when tasks didn't lend themselves to validation by histogram, categorical attributes enabled more systematic inspection. For example, P10 wrote an operation to extract a list of quotes containing logical fallacies from each debate transcript, then added a `fallacy_type` attribute to label each quote (e.g., strawman, ad hominem). Subsequent grouping by type made it easier for P10 to spot errors, since the LLM performed better on some fallacies than others. Some participants developed more sophisticated approaches; e.g., P5 requested the AI chatbot assistant to automatically generate a taxonomy of document types to use in a `map` operation, and P9 created a hierarchical taxonomy themselves and edited their prompt to include this taxonomy.

6.2 Users Iteratively Refine Pipelines to Navigate All Gulfs

Unlike typical data science workflows where users begin with exploratory data analysis [57], all participants skipped manual document review and jumped straight into writing `map` operations. As they inspected outputs, they frequently revised their pipelines in response to what they saw what we call "opportunistic realignment". This realignment helped users bridge gulfs in Fig. 1 the specification gulf, by refining prompts and operations to better express their intent; and the generalization gulf, by tuning their workflows to improve alignment with the quirks of their specific data.

Figure 8: Participants converted open-ended prompts into classification tasks for easier debugging. Here, P1 used a binary version (v1) that yielded uninformative results ("100% true"), then switched to a 5-point scale (v2) for higher accuracy. P1 could then apply a code-based rule (e.g., score >4) to interpret the results as a binary outcome.

Realignment occurred when users discovered either limitations in what the LLM could do, prompting debugging or operation decomposition, or possibilities that surfaced through surprisingly useful outputs. In both cases, users adjusted their goals or prompts based on what they learned from the system's behavior. Interestingly, even though the "Operation Decomposition" feature was designed to help address the generalization gulf, users sometimes adopted it as a way to improve specification too using suggestions to rethink how they framed their tasks or restructure their prompts.

6.2.1 LLM limitations lead users to reframe their goals. Participants often encountered LLM limitations that forced them to pursue alternate ways of accomplishing the same high-level task (P1, P4, P6, P9). These alternate paths often emerged after inspecting outputs and realizing the LLM misunderstood the original intent. For instance, in a medical information extraction task, P4 wanted to extract unique symptoms that patients mentioned in the conversation transcripts. However, the LLM frequently returned near-duplicate entries like "pain when pressure applied," "pain when lying down," and "pain when sleeping." P4 instead had wanted a small set of unique symptoms, grouping related variants in parentheses, e.g., "pain (when pressure applied, when lying down)". P6 faced a similar issue in a political debate analysis task. Their initial prompt produced long-winded topic summaries that buried the main points. They hadn't originally planned to constrain output length, but after reviewing the verbose outputs, realized they preferred more concise descriptions and added explicit brevity instructions.

In realigning with the LLM's interpretation of the pipeline, users discovered limitations in common prompt engineering strategies (P1, P4). P1 initially wanted to include successful output examples (and corresponding documents) in their prompts [82], but realized that the documents contained too much irrelevant information, making it difficult for the LLM to infer what made these outputs successful. P1 spent considerable time reasoning about the LLM's inferential capabilities, repeatedly shifting between output checking, providing notes, and prompt refinement disrupting flow [87].

Some participants redistributed work between semantic (LLM) and code-based components (P2, P4). When analyzing doctor-patient transcripts, P2 and P4 initially used a semantic map-reduce pipeline to extract and summarize symptoms. However, the reduce operation often overgeneralized, collapsing distinct mentions like knee pain and back pain into simply pain. To preserve symptom specificity, they replaced the semantic reduce with a code-based reduce that used string matching to tally each extracted symptom (maintaining distinctions between similar symptoms like knee pain and pain in the knee that would be counted as separate items even though they represent the same underlying condition). P4 then added a second map operation that used an LLM to generate a comprehensive report incorporating both the raw symptom extractions and their frequency counts giving the LLM the discretion to merge related symptoms while using the frequency data to highlight which symptoms were most common across the dataset.

Over time, participants effectively navigated the specification gulf (Fig. 1) through iteration. P9 reflected that their process was surprisingly systematic: *more like iteration and problem articulation [based on what the LLM can do], not trying random things.* In this way, LLM interaction became a medium for clarifying analytical goals echoing Schön's notion of reflection-in-action [68].

6.2.2 Unexpected possibilities can also shift analytical goals. Not all pivots were due to limitations. Some users shifted direction after spotting surprising or useful patterns in the LLM's outputs. These serendipitous findings weren't requested explicitly, but appeared occasionally, revealing new opportunities for analysis. For example, P9 initially wanted to extract symptoms from doctor-patient transcripts, but noticed that the LLM also surfaced relevant patient history for a few outputs. They found this helpful it both improved LLM accuracy and made outputs easier to validate so they updated the prompt to also extract medical history. Others (P6, P10) discovered desirable output structures they hadn't asked for like bulleted lists or Markdown formatting and revised their prompts to consistently request those formats.

Participants also encountered meaningful patterns or categories in the data that they hadn't explicitly asked for (P1, P5, P7-P10). For instance, while analyzing public safety records, P5 noticed that some outputs described the roles of people involved in each incident. They realized these roles could reveal the type of document such as a witness statement versus a legal document and used this insight to refine their pipeline. P9 became aware of this emerging process and coined the term *prompt rubber ducking* to describe how interacting with LLMs helped them figure out what questions to ask about their data. In this way, semantic data processing pipelines don't just answer predefined questions, they also help shape users' understanding of *what* questions are worth asking perhaps similar to the *berry picking* model of information seeking, where users iteratively refine their search as they gain new insights [6].

Despite frequent shifts in focus (e.g., inspecting, rejecting, refining), users maintained a strong sense of progress throughout. This stemmed from two factors. First, system responsiveness allowed rapid iteration, impressing users (P1, P3-P5, P8, P9). For example, P8 expanded a GDPR compliance analysis to include CCPA patterns simply by changing *GDPR* to *CCPA* in the prompt, and results appeared within seconds. Second, output schemas acted as what P8 described as *speed breaks*, slowing exploration just enough for meaningful reflection. We observed that several participants (P2, P6, P7) visibly slowed down when writing these schemas. Output schemas served as *semantic type checks* a form of validation analogous to type checks in traditional programming [14].

6.2.3 Decomposition Actually Supports Both Generalization and Comprehension. Although the Operation Decomposition feature was originally designed to address the generalization gulf in Fig. 1 by helping users improve output accuracy across diverse documents participants also used it to navigate the comprehension gulf, helping them understand how the LLM interpreted their data. The decomposition analysis inspired alternate task explorations, temporarily unblocking the user (P2, P4, P5, P9). All participants reviewed decomposition suggestions at least once, but only some (P2 P4, P8, P10) allowed DocWrangler to automatically apply them. A key factor in whether participants accepted suggestions was their confidence in implementing the changes on their own. When users felt capable of restructuring an operation, they preferred to use decomposition insights as inspiration either to manually rewrite prompts or provide In-Situ User Notes (P2, P3). But when the suggestions felt too complex or outside their expertise, they opted into automatic decomposition (e.g., P5 for entity resolution, or P4 when they wanted the system to do it better). P4 explicitly invoked decomposition when they could not articulate the issue as a pipeline refinement, explaining *I feel like there's a gap in me understanding how to reconcile what's being suggested and what's being set up.*

Importantly, even when participants chose to implement changes themselves rather than accepting automatic decomposition, they still benefited from DocWrangler's ability to identify issues they might have missed. For example, while manually verifying every extraction across all documents was impractical, it was much easier for participants confirm whether the decomposer's specific notifications about missed information were accurate (P4, P9). AI-assisted analysis, in this way, helps users bridge the comprehension gulf by pinpointing specific data they might otherwise overlook.

6.3 Participants Identified Gaps in Tooling

Participants pointed out specific limitations in DocWrangler that made it harder to interpret or reason about LLM outputs. These gaps often came up as feature requests, highlighting needs for better support with pattern discovery, provenance tracing, and operation-specific visualization.

The Necessity of Bottom-up Validation. All participants recognized that they were spending a lot of time validating outputs, and some explicitly requested to automate this with LLM-as-a-judge approaches (P4, P7). Rather than defining evaluation criteria upfront or using a top-down approach, as prior work suggests [39, 51, 72], participants (P2, P4) preferred to discover criteria bottom-up through hands-on review similar to qualitative coding [38], but prompted by LLMs identifying both interesting patterns and outliers. P4 considered writing an operation to validate the previous operation and began thinking of a rubric to put in the prompt, but realized they'd need to review many outputs first to come up with this rubric, and then repeat this process after each pipeline change. With goals shifting frequently, this felt impractical. Others (P1, P2) requested tools

for organic pattern discovery, such as P1's request for side-by-side comparisons of LLM outputs across diverse, automatically-selected documents, to more easily spot issues and provide in-situ notes.

Additional Support for Inspecting Provenance. Participants wanted easier ways to trace how LLM outputs were derived from the source text without having to manually search documents or write custom checks (e.g., a code-based filter after each map). Some (P2, P8, P9) requested that outputs be directly linked to their source text, with the relevant span highlighted. However, simple checks for the presence of terms don't guarantee the operation was performed correctly. For example, the LLM listed "muscle aches" as a symptom, even though it appeared only in the doctor's question not as something the patient reported, showing how surface-level matches can be misleading (P9). Moreover, users wanted DocWrangler to trace errors to their source. For example, when P4 reviewed extracted medication information, there were incorrect dosages in the source text (e.g., "200g" instead of "200mg") that the LLM faithfully reproduced. This highlights how LLM pipelines blur the boundary between data cleaning and analysis, unlike traditional workflows where these phases are less intertwined [36, 57]. We never observed participants creating dedicated cleaning operations perhaps, as P2 mentioned, because they expected LLMs to implicitly clean data (e.g., recognize that "200g" is not a valid medication dose).

Operation-Specific Visualization Tools. Users requested richer visualizations beyond basic histograms and bar charts (P4, P5, P7, P10), with needs varying by operation type and data domain. P7 requested LLM-generated custom charts. P4 asked for DocWrangler to track output distribution changes between iterations, particularly when using map operations as classifiers (Section 6.1.2).

7 Real-World Deployment and Usage

To complement our qualitative user study, we deployed DocWrangler as a public web application, collecting telemetry data from over 1,500 pipeline executions across two months. We used DocWrangler itself to analyze this telemetry data, manually verifying a sample of 50 extractions and classifications.[4] We will discuss the domains and document types in which DocWrangler was used, common task patterns and pipeline structures, how pipelines evolved, and how users engaged with AI assistance features.

Pipeline Data, Task, and Model Patterns. DocWrangler was used across a range of professional domains. Common applications included legal (e.g., contract clause extraction), healthcare (e.g., analyzing clinical case studies), finance (e.g., parsing invoices and budgets), and education (e.g., generating test questions from textbooks). Other frequent use cases involved customer feedback analysis, government document processing, and media or news content analysis. About 50% of documents were semi-structured with hierarchical sections, though often inconsistently formatted. Unstructured text was the next most common, typically in PDF format. Pipelines spanned 9 languages: English, Spanish, Chinese, German, Russian, Japanese, Italian, Greek, and Persian.

Most pipelines centered on a few key task types. Extraction was most common, appearing in over half of all pipelines from pulling out existing structure to identifying semantically meaningful entities in the absence of structure. Classification ranked second, typically categorizing documents or content. Summarization usually followed these steps using reduce operations. Most workflows followed a simple pattern: extract, classify, then summarize, but some domains exhibited different patterns. For example, in business analysis, users built complex pipelines (sometimes with 15+ operations) that performed sequential extractions—first identifying entities like departments and processes, then mapping relationships, and finally generating reports. In the legal domain, pipelines were often multi-step as well, but focused on extracting varied or less-related entities, such as claims and case law references. For research domains, pipelines often involved more open-ended pattern discovery across documents with map operations then more structured map-reduce pipelines. Most pipelines were simple: 75% had two or fewer operations, and 90% had no more than three. But 5% had more than five operations, and a few pipelines exceeded 30, reflecting highly customized workflows.

Pipelines used different AI models. gpt-4o-mini (our default model) was the most common model, used in 82% of pipelines. gpt-4o appeared in 12% of pipelines, typically for more complex workflows or in combination with gpt-4o-mini. Gemini models were used in 17%, often for PDF processing. Claude-3.5-sonnet showed up in 9%, mainly for classification. 18% of pipelines used multiple models.

Pipeline Evolution Patterns. We analyzed how users modified their pipelines over time by examining consecutive pipeline versions created within 5-minute intervals. We observed three main evolution patterns: 53% of pipelines grew more complex by adding operations or upgrading models; 18% actually became simpler through operation consolidation or reduced sample sizes; and 29% maintained the same operations while only changing prompts or output schemas. We also tracked how prompts evolved: 47% became more specific with structured schemas and domain-specific language; 16% moved toward simpler prompts (following the "prompt rubber ducking" strategy mentioned in Section 6.2.2); and 37% maintained similar detail levels with minor refinements.

AI Assistance and Challenges. We tracked AI assistance features: prompt improvement (150 instances) and general AI chat (95 instances). Surprisingly, for prompt refinement, over half of these cases involved users refining their prompts without providing any notes. Users were invoking prompt refinement proactively to transform their vague, poorly-specified instructions into more concrete and executable prompts; for example, one user invoked prompt refinement to transform a prompt like "Extract the information from the ledger" into a detailed 10-line specification identifying precise fields to extract and noting important context, such as these are scans of handwritten documents. This reveals how users often struggle with formulating effective prompts from the outset. Through the AI chat, users asked general questions about syntax, workflow creation, and supported file formats. Some users still expressed frustration when their prompts didn't work as expected, with notes like "MAP THEM ALL SYSTEMATICALLY!!!" and "WHY IS THIS SO HARD?!" These challenges suggest we need better guidance and clearer explanations of DocWrangler's capabilities and limitations, particularly for new users.

[4] Our analysis involved three map-reduce pipelines with gpt-4o; one for each topic (e.g., usage domains and document types). Our total analysis cost was $9.39 USD.

8 Discussion

Here, we discuss what our findings mean for current data processing systems and human-AI collaboration more broadly.

Designing Better Semantic Data Processing Tools. DocWrangler is an early attempt at building an IDE for semantic data processing, which taught us a lot about how users interact with data analysis workflows with LLM-powered operators. Based on our findings, we reflect on opportunities for future semantic data processing systems.

When building DocWrangler, we realized that bridging the specification gulf involves two different challenges: finding the right question to ask (i.e., intent discovery) and clearly expressing that question in the pipeline (i.e., refining ambiguity). While we tried to address both with the same features in DocWrangler, they are fundamentally different. Discovering intent is exploratory: users need to understand what's interesting or important in their data before formulating precise questions. For structured data, we already have effective tools for kick-starting exploration, by automatically discovering and surfacing typical patterns or outliers [44, 76, 95] or by letting users search for specific patterns through sketching or demonstration [7, 30, 77]. However, these approaches need reimagining for unstructured data and semantic processing. It's not clear what constitutes a "document anomaly," or how to tell whether documents are meaningfully different in size, structure, or content. Future systems could address this challenge and help proactively highlight patterns in the underlying documents, ultimately making it easier for humans to decide what they want to query. In contrast, clarifying intent once identified is about precise specification and addressing ambiguity, and may be more amenable to automation. Systems can suggest more precise wording and recommend better alignment with how LLMs interpret instructions (similar to how we supported prompt refinement in DocWrangler).

Additionally, our study revealed several debugging strategies that future systems could partially automate. For example, users frequently transformed unstructured extraction tasks into more structured ones to make validation easier (Section 6.1.2) so, systems could detect when outputs lack structure and automatically suggest complementary structured attributes to aid validation. Similarly, when systems detect skewed output distributions (like P1's all-"true" trust classifications in Fig. 8), they could automatically rewrite prompts to yield more useful variations. Systems could also automatically generate rationales as interpretability tools (Section 6.1.1), by providing explanations, clustering or finding anomalous examples, and presenting these for review to help users spot intent-interpretation mismatches.

Overall, semantic data processing, and the advent of LLMs, has the potential to change how we build the next generation of data systems [21] that combine both structured and unstructured data processing capabilities. Our systems are no longer passive executors of fixed specifications like traditional data processing tools; instead, they must help users express and refine semantic needs through their pipelines [102]. To make semantic data processing truly usable, we must design systems that actively bridge all three gulfs between users' intentions, pipeline specifications, and underlying data be it in a structured or unstructured context.

Semantic Data Processing as a Lens on Human-AI Collaboration. Semantic data processing serves as a rich domain to study effective human-AI collaboration. The interaction triangle shown in Fig. 1 may generalize to other human-AI systems if we recognize that "data" might be much smaller or take different forms than document collections (e.g., a single essay, a piece of software). Consequently, our findings might offer insights for designing effective human-AI systems across domains.

First, we reflect on *how* to design AI systems. We observed users creating what creativity support tool (CST) research calls epistemic artifacts [89]—exploratory objects that help users discover possibilities. Early pipeline iterations served as epistemic artifacts, helping users learn about their data rather than being final solutions, similar to how artists explore materials before creating finished works. Though DocWrangler wasn't intended to be a CST, it functioned as one because tasks had inherent fuzziness, and users learned about either their preferences or the task itself through LLMs. In this way, any system addressing ambiguous tasks may benefit from CST design principles; e.g., supporting exploration without predefined goals and allowing movement between different levels of abstraction [46, 89].

Then, like CSTs, AI systems must preserve user agency[5] [31]. To maintain this agency, prior work suggests using shared representations of information that both humans and AI can modify [27], or creating different representations that give users varying levels of control [67]. Our work adds another perspective: creating moments for user reflection [68]. For example, our in-situ notes feature allowed (perhaps even encouraged, due to its ubiquitous presence) users to record thoughts. We also unintentionally created reflective spaces: e.g., output schema definitions slowed users down, prompting their reflection on their goals (P8).

We also reflect on *what* to design for AI systems. Each gulf in Fig. 1 represents a distinct design challenge, with multiple possible approaches to address each one (not just what DocWrangler proposes). Moreover, applications should offer features to reduce the burden of navigating all gulfs, but specialized domains may benefit from specialized approaches. For instance, AI image generators already bridge the specification gulf with controls for attributes like realism, style, and composition [10, 56, 66], but these systems can also address the generalization gulf by automatically aging semantic inconsistencies like anatomical errors or impossible scenes constraints users shouldn't need to explicitly state if they want high realism. In another example, an AI coding assistant could use compilation errors to self-refine its generated code [62] (bridging the generalization gulf), while also maintaining these errors as notes and summarizing them (aiding the comprehension gulf). Or, consider an AI vacation planner: interactive visualizations of possible trips could help users comprehend both the recommendations and what the AI has inferred about their preferences.

Overall, based on our work, we present the following recommendations for building effective human-AI collaborative systems: (i) design for epistemic artifacts that help users explore and understand their problem space, (ii) create intentional moments for reflection that maintain user agency amidst rapid AI-driven iterations; and

[5] We acknowledge that "agency" has been critiqued as an overly broad term in HCI [8]; here we also use it in this general sense, rather than referring to a specific definition.

(iii) develop features that explicitly address each of the three gulfs speci cation, generalization, and comprehension recognizing that bridging one gulf can indirectly help with others.

Limitations. We re ect on some limitations of our work. First, our participants were all tech-savvy and had used LLM tools before, with most having programming experience (though three had not heard of map-reduce before). This may have a ected how easily they adapted to our system. Future work should include users with varying technical backgrounds. Second, we only observed sessions lasting 1-2 hours, while real document analysis often spans days or weeks. Longitudinal studies could reveal how behavior evolves over longer periods and in team settings. In our online deployment, as users needed to provide their own API keys for LLMs, our user base is self-selecting primarily comprising individuals who discovered DocWrangler through blog posts and technical talks. Additionally, some users, particularly those with more experience, run DocWrangler on their own infrastructure, for which we lack telemetry data. Finally, our ndings come from observing users with our speci c system. While we believe the patterns we iden- ti ed re ect fundamental aspects of human-AI interaction, more research is needed to understand how they manifest across di erent interfaces, LLM capabilities, and domains.

9 Conclusion

This paper introduced DocWrangler , a mixed-initiative IDE for semantic data processing, where LLMs power familiar data operations like map and reduce on unstructured text. We contributed three novel features to bridge the gulfs between users, their data, and their semantic data pipelines: in-situ user notes, LLM-assisted prompt re nement, and LLM-assisted operation decomposition. Our evaluation through both a qualitative user study and large-scale online deployment provides insights into how people adapt to and use DocWrangler for their tasks; particularly, how they write pipelines for the purpose of learning more about their data or LLM capabilities. Finally, our work provides a re ection on how data systems could be redesigned for the LLM era and what semantic data processing can teach us about e ective human-AI collaboration more broadly.

Acknowledgments

We are grateful to Shm Almeda, Ian Arawjo, Timothy Aveni, Quentin Romero Lauro, Yiming Lin, HC Moore, James Smith, J.D. Zam rescu-Pereira, and Sepanta Zeighami for their thoughtful feedback and valuable discussions on our ndings. Special thanks to our study participants, whose insights and engagement were instrumental in understanding how users interact with semantic data processing systems. This work was supported by the National Science Foundation (grants DGE-2243822, IIS-2129008, IIS-1940759, IIS-1940757, 1845638, 1740305, 2008295, 2106197, 2103794, 2312991), funds from the State of California, funds from the Alfred P. Sloan Foundation, and EPIC lab sponsors (G-Research, Adobe, Microsoft, Google, Sigma Computing). Additional support was provided by Amazon and CAIT. We thank Modal Labs for their generous compute credits. SS is supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

References

[1] Shm Garanganao Almeda, JD Zam rescu-Pereira, Kyu Won Kim, Pradeep Mani Rathnam, and Bjoern Hartmann. 2024. Prompting for Discovery: Flexible Sense-Making for AI Art-Making with Dreamsheets. In Proceedings of the CHI Conference on Human Factors in Computing Systems. 1–17.

[2] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N Bennett, Kori Inkpen, et al. 2019. Guidelines for human-AI interaction. In Proceedings of the 2019 chi conference on human factors in computing systems. 1–13.

[3] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. [n. d.]. Guidelines for Human-AI Interaction. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (Glasgow Scotland Uk, 2019-05-02). ACM, 1–13. https://doi.org/10.1145/3290605.3300233

[4] Eric Anderson, Jonathan Fritz, Austin Lee, Bohou Li, Mark Lindblad, Henry Lindeman, Alex Meyer, Parth Parmar, Tanvi Ranade, Mehul A. Shah, Benjamin Sowell, Dan Tecuci, Vinayak Thapliyal, and Matt Welsh. [n. d.]. The Design of an LLM-powered Unstructured Analytics System. arXiv:2409.00847 [cs] http://arxiv.org/abs/2409.00847

[5] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L Glassman. 2024. ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing. In Proceedings of the CHI Conference on Human Factors in Computing Systems. 1–18.

[6] Marcia J Bates. 1989. The design of browsing and berrypicking techniques for the online search interface. Online review 13, 5 (1989), 407–424.

[7] Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic prefetching of data tiles for interactive visualization. In Proceedings of the 2016 International Conference on Management of Data. 1363–1375.

[8] Dan Bennett, Oussama Metatla, Anne Roudaut, and Elisa D Mekler. 2023. How does HCI understand human agency and autonomy? In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. 1–18.

[9] Alan Blackwell and Thomas Green. 2003. Notational systems the cognitive dimensions of notations framework. HCI models, theories, and frameworks: toward an interdisciplinary science. Morgan Kaufmann 234 (2003).

[10] Stephen Brade, Bryan Wang, Mauricio Sousa, Sageev Oore, and Tovi Grossman. 2023. Promptify: Text-to-image generation through interactive prompt exploration with large language models. In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology. 1–14.

[11] Bazyli Brzoska. 2025. gpt-tokenizer: A Token Byte Pair Encoder/Decoder for OpenAI Models. https://www.npmjs.com/package/gpt-tokenizer Accessed: 2025-04-07.

[12] Valerie Chen, Alan Zhu, Sebastian Zhao, Hussein Mozannar, David Sontag, and Ameet Talwalkar. 2024. Need Help? Designing Proactive AI Assistants for Programming. arXiv preprint arXiv:2410.04596 (2024).

[13] Bhavya Chopra, Ananya Singha, Anna Fariha, Sumit Gulwani, Chris Parnin, Ashish Tiwari, and Austin Z Henley. 2023. Conversational challenges in ai-powered data science: Obstacles, needs, and design opportunities. arXiv preprint arXiv:2310.16164 (2023).

[14] Michael Coblenz, Chris Martens, and Luke Church. 2021. Programming Languages + Human-Computer Interaction: Continuing the story at SPLASH 2020. SIGPLAN Blog. https://blog.sigplan.org/2021/07/06/programming-languages-human-computer-interaction-continuing-the-story-at-splash-2020/ Accessed: 2025-04-03.

[15] Mary Czerwinski, Eric Horvitz, and Susan Wilhite. 2004. A diary study of task switching and interruptions. In Proceedings of the SIGCHI conference on Human factors in computing systems. 175–182.

[16] Hai Dang, Lukas Mecke, Florian Lehmann, Sven Goller, and Daniel Buschek. 2022. How to prompt? Opportunities and challenges of zero-and few-shot learning for human-AI interaction in creative applications of generative models. arXiv preprint arXiv:2209.01390 (2022).

[17] Je rey Dean and Sanjay Ghemawat. 2008. MapReduce: simpli ed data processing on large clusters. Commun. ACM 51, 1 (2008), 107–113.

[18] AnHai Doan, Je rey F Naughton, Raghu Ramakrishnan, Akanksha Baid, Xiaoyong Chai, Fei Chen, Ting Chen, Eric Chu, Pedro DeRose, Byron Gao, et al. 2009. Information extraction challenges in managing unstructured data. ACM SIGMOD Record 37, 4 (2009), 14–20.

[19] John J. Dudley and Per Ola Kristensson. [n. d.]. A Review of User Interface Design for Interactive Machine Learning. 8, 2 ([n. d.]), 1–37. https://doi.org/10.1145/3185517

[20] KJ Feng, Kevin Pu, Matt Latzke, Tal August, Pao Siangliulue, Jonathan Bragg, Daniel S Weld, Amy X Zhang, and Joseph Chee Chang. 2024. Cocoa: Co-Planning and Co-Execution with AI Agents. arXiv preprint arXiv:2412.10999 (2024).

[21] Raul Castro Fernandez, Aaron J Elmore, Michael J Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How large language models will disrupt data management. Proceedings of the VLDB Endowment 16, 11 (2023), 3302–3309.

[22] Katy Ilonka Gero, Chelse Swoopes, Ziwei Gu, Jonathan K Kummerfeld, and Elena L Glassman. 2024. Supporting Sensemaking of Large Language Model Outputs at Scale. In Proceedings of the CHI Conference on Human Factors in Computing Systems 1 21.

[23] Madeleine Grunde-McLaughlin, Michelle S. Lam, Ranjay Krishna, Daniel S. Weld, and Je rey Heer. [n. d.]. Designing LLM Chains by Adapting Techniques from Crowdsourcing Work ows. arXiv:2312.11681 [cs] http://arxiv.org/abs/2312.11681

[24] Ken Gu, Madeleine Grunde-McLaughlin, Andrew McNutt, Je rey Heer, and Tim Altho . 2024. How do data analysts respond to ai assistance? a wizard-of-oz study. In Proceedings of the CHI Conference on Human Factors in Computing Systems 1 22.

[25] Sumit Gulwani. 2016. Programming by examples: Applications, algorithms, and ambiguity resolution. In Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 July 2, 2016, Proceedings 8. Springer, 9 14.

[26] Hossein Hassani and Emmanuel Sirmal Silva. 2023. The role of ChatGPT in data science: how ai-assisted conversational interfaces are revolutionizing the eld. Big data and cognitive computing 7, 2 (2023), 62.

[27] Je rey Heer. 2019. Agency plus automation: Designing arti cial intelligence into interactive systems. Proceedings of the National Academy of Sciences 116, 6 (2019), 1844 1850.

[28] Je rey Heer, Joseph M Hellerstein, and Sean Kandel. 2015. Predictive Interaction for Data Transformation.. In CIDR. Citeseer.

[29] Joseph M Hellerstein, Michael Stonebraker, James Hamilton, 2007. Architecture of a database system. Foundations and Trends in Databases 1, 2 (2007), 141 259.

[30] Harry Hochheiser and Ben Shneiderman. 2004. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. Information Visualization 3, 1 (2004), 1 18.

[31] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems. 159 166.

[32] Gary W Johnson and Richard Jennings. 2001. LabVIEW graphical programming. McGraw-Hill Professional.

[33] Adam Tauman Kalai and Santosh S Vempala. 2024. Calibrated language models must hallucinate. In Proceedings of the 56th Annual ACM Symposium on Theory of Computing. 160 171.

[34] Sean Kandel, Je rey Heer, Catherine Plaisant, Jessie Kennedy, Frank Van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. 2011. Research directions in data wrangling: Visualizations and transformations for usable and credible data. Information Visualization 10, 4 (2011), 271 288.

[35] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Je rey Heer. 2011. Wrangler: Interactive visual speci cation of data transformation scripts. In Proceedings of the sigchi conference on human factors in computing systems. 3363 3372.

[36] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Je rey Heer. 2012. Enterprise data analysis and visualization: An interview study. IEEE transactions on visualization and computer graphics 18, 12 (2012), 2917 2926.

[37] Majeed Kazemitabaar, Jack Williams, Ian Drosos, Tovi Grossman, Austin Zachary Henley, Carina Negreanu, and Advait Sarkar. 2024. Improving steering and veri cation in AI-assisted data analysis with interactive task decomposition. In Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology. 1 19.

[38] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. [n. d.]. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. arXiv:2310.03714 [cs] http://arxiv.org/abs/2310.03714

[39] Tae Soo Kim, Yoonjoo Lee, Jamin Shin, Young-Ho Kim, and Juho Kim. 2024. Evallm: Interactive evaluation of large language model prompts on user-de ned criteria. In Proceedings of the CHI Conference on Human Factors in Computing Systems 1 21.

[40] David Kirsh and Paul Maglio. 1994. On distinguishing epistemic from pragmatic action. Cognitive science 18, 4 (1994), 513 549.

[41] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks a publishing format for reproducible computational work ows. In Positioning and power in academic publishing: Players, agents and agendas. IOS press, 87 90.

[42] Michelle S Lam, Janice Teoh, James A Landay, Je rey Heer, and Michael S Bernstein. 2024. Concept Induction: Analyzing Unstructured Text with High-Level Concepts Using LLooM. In Proceedings of the CHI Conference on Human Factors in Computing Systems 1 28.

[43] Doris Jung Lin Lee and Aditya G Parameswaran. 2018. The Case for a Visual Discovery Assistant: A Holistic Solution for Accelerating Visual Data Exploration. IEEE Data Eng. Bull. 41, 3 (2018), 3 14.

[44] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A Hearst, et al. 2021. Lux: always-on visualization recommendations for exploratory dataframe work ows. arXiv preprint arXiv:2105.00120 (2021).

[45] Arthur Li. 2013. Handbook of SAS DATA Step Programming. CRC press.

[46] Jingyi Li, Eric Rawn, Jacob Ritchie, Jasper Tran O'Leary, and Sean Follmer. 2023. Beyond the artifact: power as a lens for creativity support tools. In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology 1 15.

[47] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. arXiv preprint arXiv:2405.14696 (2024).

[48] Chunwei Liu, Gerardo Vitagliano, Brandon Rose, Matt Prinz, David Andrew Samson, and Michael Cafarella. 2025. PalimpChat: Declarative and Interactive AI analytics. arXiv preprint arXiv:2502.03368 (2025).

[49] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. Transactions of the Association for Computational Linguistics 12 (2024), 157 173.

[50] Vivian Liu and Lydia B Chilton. 2022. Design guidelines for prompt engineering text-to-image generative models. In Proceedings of the 2022 CHI conference on human factors in computing systems 1 23.

[51] Qianou Ma, Weirui Peng, Hua Shen, Kenneth Koedinger, and Tongshuang Wu. 2024. What you say= what you want? Teaching humans to articulate requirements for LLMs. arXiv preprint arXiv:2409.08775 (2024).

[52] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegre e, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-re ne: Iterative re nement with self-feedback. Advances in Neural Information Processing Systems 36 (2023), 46534 46594.

[53] Samuel Madden, Michael Cafarella, Michael Franklin, and Tim Kraska. 2024. Databases Unbound: Querying All of the World's Bytes with AI. Proceedings of the VLDB Endowment 17, 12 (2024), 4546 4554.

[54] Wes McKinney et al. 2011. pandas: a foundational Python library for data analysis and statistics. Python for high performance and scienti c computing 14, 9 (2011), 1 9.

[55] Andrew M McNutt, Chenglong Wang, Robert A Deline, and Steven M Drucker. 2023. On the design of ai-powered code assistants for notebooks. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems 1 16.

[56] Midjourney. 2025. Midjourney: AI-based Image Generation Service. https://www.midjourney.com/. Accessed: 2025-04-09.

[57] Michael Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q Vera Liao, Casey Dugan, and Thomas Erickson. 2019. How data science workers work with data: Discovery, capture, curation, design, creation. In Proceedings of the 2019 CHI conference on human factors in computing systems 1 15.

[58] Tamara Munzner. 2014. Visualization analysis and design. CRC press.

[59] Allen Newell. 1972. Human problem solving. Upper Saddle River/Prentive Hall (1972).

[60] Jakob Nielsen. 1994. Usability engineering. Morgan Kaufmann.

[61] Tobias Nipkow. 2003. Jinja: Towards a comprehensive formal semantics for a Java-like language. In Proc. Marktobberdorf Summer School. IOS Press Amsterdam.

[62] D. A. Norman. 1987. Some observations on mental models. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 241 244.

[63] Aditya Parameswaran, Shreya Shankar, Parth Asawa, Naman Jain, and Yujie Wang. 2023. Revisiting Prompt Engineering via Declarative Crowdsourcing. (2023). https://par.nsf.gov/biblio/10531530

[64] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. Lotus: Enabling semantic queries with llms over tables of unstructured and structured data. arXiv preprint arXiv:2407.11418 (2024).

[65] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In VLDB, Vol. 1. 381 390.

[66] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent di usion models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition 10684 10695.

[67] Arvind Satyanarayan. 2024. Intelligence as Agency. Adjunct Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology 1 3.

[68] Donald A Schön. 1979. The re ective practitioner. New York (1979).

[69] Vidya Setlur and Melanie Tory. 2022. How do you converse with an analytical chatbot? revisiting gricean maxims for designing analytical conversational behavior. In Proceedings of the 2022 CHI conference on human factors in computing systems 1 17.

[70] Shreya Shankar, Haotian Li, Parth Asawa, Madelon Hulsebos, Yiming Lin, J Zam rescu-Pereira, Harrison Chase, Will Fu-Hinthorn, Aditya G Parameswaran, and Eugene Wu. 2024. SPADE: Synthesizing Data Quality Assertions for Large Language Model Pipelines. Proc. VLDB Endow. (2024).

[71] Shreya Shankar, Aditya G. Parameswaran, and Eugene Wu. 2024. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. arXiv:2410.12189 [cs.DB] https://arxiv.org/abs/2410.12189

[72] Shreya Shankar, JD Zam rescu-Pereira, Björn Hartmann, Aditya Parameswaran, and Ian Arawjo. 2024. 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. 1 14.

[73] Ben Shneiderman. 2003. The eyes have it: A task by data type taxonomy for information visualizations. In The craft of information visualization. Elsevier, 364 371.

[74] Ben Shneiderman. 2020. Human-centered arti cial intelligence: Reliable, safe & trustworthy. International Journal of Human Computer Interaction 36, 6 (2020), 495 504.

[75] Nischal Shrestha, Titus Barik, and Chris Parnin. 2021. Unravel: A uent code explorer for data wrangling. In The 34th Annual ACM Symposium on User Interface Software and Technology. 198 207.

[76] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. 2016. E ortless Data Exploration with zenvisage: An Expressive and Interactive Visual Analytics System. Proceedings of the VLDB Endowment 10, 4 (2016).

[77] Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, and Aditya Parameswaran. 2018. Shapesearch: exible pattern-based querying of trend line visualizations. Proceedings of the VLDB Endowment 11, 12 (2018).

[78] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more e ective than scaling model parameters. arXiv preprint arXiv:2408.03314 (2024).

[79] Robert Soden, Laura Devendorf, Richmond Y Wong, Lydia B Chilton, Ann Light, and Yoko Akama. 2020. Embracing uncertainty in HCI. Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems. 1 8.

[80] Inc. Software Arts. 1979. VisiCalc. Computer software. https://www.bricklin.com/visicalc.htm Originally developed by Dan Bricklin and Bob Frankston.

[81] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. IEEE Transactions on visualization and computer graphics 8, 1 (2002), 52 65.

[82] Hendrik Strobelt, Albert Webson, Victor Sanh, Benjamin Hoover, Johanna Beyer, Hanspeter P ster, and Alexander M Rush. 2022. Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. IEEE transactions on visualization and computer graphics 29, 1 (2022), 1146 1156.

[83] Hari Subramonyam, Roy Pea, Christopher Pondoc, Maneesh Agrawala, and Colleen Seifert. 2024. Bridging the Gulf of Envisioning: Cognitive Challenges in Prompt Based Interactions with LLMs. In Proceedings of the CHI Conference on Human Factors in Computing Systems. 1 19.

[84] Sangho Suh, Bryan Min, Srishti Palani, and Haijun Xia. 2023. Sensecape: Enabling multilevel exploration and sensemaking with large language models. In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology. 1 18.

[85] Annalisa Szymanski, Noah Ziems, Heather A Eicher-Miller, Toby Jia-Jun Li, Meng Jiang, and Ronald A Metoyer. 2025. Limitations of the LLM-as-a-Judge approach for evaluating LLM outputs in expert knowledge tasks. Proceedings of the 30th International Conference on Intelligent User Interfaces. 952 966.

[86] Alex Tamkin, Kunal Handa, Avash Shrestha, and Noah Goodman. 2022. Task ambiguity in humans and language models. arXiv preprint arXiv:2212.10711 (2022).

[87] Lev Tankelevitch, Viktor Kewenig, Auste Simkute, Ava Elizabeth Scott, Advait Sarkar, Abigail Sellen, and Sean Rintel. 2024. The metacognitive demands and opportunities of generative AI. In Proceedings of the CHI Conference on Human Factors in Computing Systems. 1 24.

[88] Steven J Taylor, Robert Bogdan, and Marjorie L DeVault. 2015. Introduction to qualitative research methods: A guidebook and resource. John Wiley & Sons.

[89] Martin Tricaud and Michel Beaudouin-Lafon. 2023. Revisiting creative behaviour as an epistemic process: lessons from 12 computational artists and designers. In Proceedings of the 35th Australian Computer-Human Interaction Conference. 175 190.

[90] VERBI Software. 2022. MAXQDA 2022. Berlin, Germany. https://www.maxqda.com

[91] Jiayi Wang and Guoliang Li. 2025. Aop: Automated and interactive llm pipeline orchestration for answering complex queries. CIDR.

[92] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. ACM computing surveys (csur) 53, 3 (2020), 1 34.

[93] Colin Ware. 2019. Information visualization: perception for design. Morgan Kaufmann.

[94] Michael Williams and Tami Moser. 2019. The art of coding and thematic exploration in qualitative research. International management review 15, 1 (2019), 45 55.

[95] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Je rey Heer. 2015. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. IEEE transactions on visualization and computer graphics 22, 1 (2015), 649 658.

[96] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In Proceedings of the 2022 CHI conference on human factors in computing systems. 1 22.

[97] Yifan Wu, Joseph M Hellerstein, and Arvind Satyanarayan. 2020. B2: Bridging code and interactive visualization in computational notebooks. In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology. 152 165.

[98] Liwenhan Xie, Chengbo Zheng, Haijun Xia, Huamin Qu, and Chen Zhu-Tian. 2024. WaitGPT: Monitoring and Steering Conversational LLM Agent in Data Analysis with On-the-Fly Code Visualization. In Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology. 1 14.

[99] Kuat Yessenov, Shubham Tulsiani, Aditya Menon, Robert C Miller, Sumit Gulwani, Butler Lampson, and Adam Kalai. 2013. A colorful approach to text processing by example. In Proceedings of the 26th annual ACM symposium on User interface software and technology. 495 504.

[100] JD Zam rescu-Pereira, Eunice Jun, Michael Terry, Qian Yang, and Björn Hartmann. 2025. Beyond Code Generation: LLM-supported Exploration of the Program Design Space. arXiv preprint arXiv:2503.06911 (2025).

[101] JD Zam rescu-Pereira, Heather Wei, Amy Xiao, Kitty Gu, Grace Jung, Matthew G Lee, Bjoern Hartmann, and Qian Yang. 2023. Herding AI cats: Lessons from designing a chatbot by prompting GPT-3. In Proceedings of the 2023 ACM Designing Interactive Systems Conference. 2206 2220.

[102] Sepanta Zeighami, Yiming Lin, Shreya Shankar, and Aditya Parameswaran. [n. d.]. LLM-Powered Proactive Data Systems. Data Engineering ([n. d.]), 90.

[103] Jingyue Zhang and Ian Arawjo. [n. d.]. ChainBuddy: An AI Agent System for Generating LLM Pipelines. arXiv:2409.13588 [cs] http://arxiv.org/abs/2409.13588

[104] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2024. Chat2Data: An Interactive Data Analysis System with RAG, Vector Databases and LLMs. Proceedings of the VLDB Endowment 17, 12 (2024), 4481 4484.

[105] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. [n. d.]. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs] http://arxiv.org/abs/2306.05685

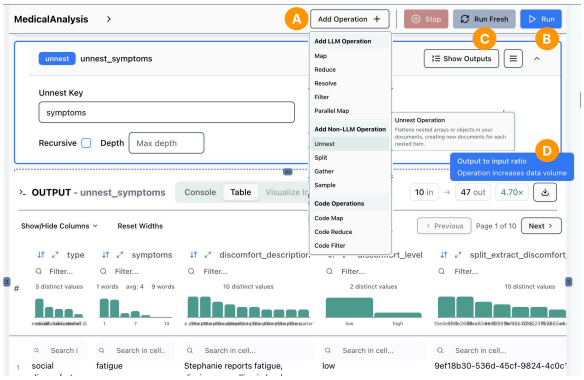

Figure 9: Adding an `unnest` operation to analyze symptoms individually. (A) User adds a new operation. (B) "Run" button executes with previously-cached operation outputs, while (C) "Run Fresh" reprocesses all operations from scratch. (D) Output viewer shows operation selectivity (10 documents expanded to 47; a 4″7× increase), with each symptom as a separate row preserving associated metadata.

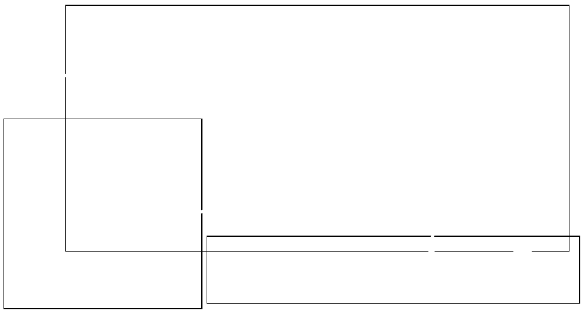

Figure 10: Adding a `reduce` operation to summarize symptom data. (A) The user adds a new operation. (B) They specify which field to group by. (C) The help menu shows the AI assistant. (D) The AI assistant helps with Jinja syntax. (E) User copies the assistant's suggestion to copy into the pipeline editor. (F) The user runs the pipeline. (G) The output includes a visualization of the distribution of symptoms.

A Extended Feature Walk-Through

Here, we continue the feature walk-through from Section 4.2, illustrating use of different operators and the AI assistant chatbot.

Recall that at the end of Section 4.2, the analyst has decomposed their first operation into the pipeline described in Fig. 6. With the decomposed pipeline now reliable, the analyst resumes analyzing discomfort by symptom. They click "Add Operation" (Fig. 9A) and select an `unnest` operation, specifying "symptoms" as the attribute to flatten.[6] After configuration, they click "Run" (Fig. 9B), which uses cached outputs from previous operations, though "Run Fresh" (Fig. 9C) remains available for complete reprocessing. The operation executes instantly, with the output viewer showing *"10 in → 47 out"* and "4″70×" (Fig. 9D). Each row now represents a single symptom, with all other data attributes (the original transcript text and LLM-extracted attributes like discomfort level) copied from the source document that contained that symptom.

To summarize discomfort patterns by symptom, the analyst adds a `reduce` operation (Fig. 10A-B), specifying "symptoms" as the group-by attribute. Unsure about the how to write a Jinja template to loop over a group of documents, they access the chat-based AI assistant (Fig. 10C-D) and request help writing their prompt. Using the AI's suggested syntax (Fig. 10E), they run the pipeline (Fig. 10F) and examine the symptom distribution (Fig. 10G). They can switch to the table view for detailed LLM output inspection as needed. Inspection—and the data analysis, in general—proceeds for as long as the user would like.

Figure 11: Prompt Refinement interface. (A) The interface visualizes the diff between prompt versions. (B) An interactive revision history tree allows users to view and branch from previous prompt versions.

[6] `unnest` is akin to the "explode" operator in Pandas.