

Agentic AI: A Case Study on the Visualization Publications Dataset

Zainab Aamir*
Stony Brook University

Saeed Boorboor†
University of Illinois Chicago

Arie E. Kaufman,‡
Stony Brook University



Figure 1: Workflow of our agentic pipeline. *Inspect Dataset*, our agent classifies the dataset and identifies candidate text fields. *Route Agent* then selects the target field based on dataset type. Next, *Analyzing Text* applies hybrid keyword extraction, embeddings, and clustering, followed by LLM-based summarization to derive concise labels. In *Plot Generation*, a two-stage process produces validated Vega–Lite specifications. Finally, *Validation and Refinement* ensures correctness, readability, and narrative coherence before assembling the final gallery.

ABSTRACT

Large Language Models (LLMs), trained on vast and diverse corpora, have demonstrated strong capabilities in reasoning across domains and generating structured outputs such as code, text, and visual encodings. However, when used in isolation, LLMs often struggle with reliability, correctness, and readability. This has motivated significant interest in agentic Artificial Intelligence (AI) systems that scaffold, validate, and refine the model’s behavior for trustworthy and autonomous decision-making. In this paper, we present an agent that inspects unfamiliar datasets, classifies their type, and routes them through tailored processing pipelines. For the VIS Publication dataset, the agent selects the *Abstract* column as the richest semantic signal of research content and performs temporal topic evolution to trace how visualization research themes emerge and shift over time. The pipeline combines deterministic text-mining components with selective LLM usage for concise cluster labeling, thereby reducing redundant token consumption while leveraging LLM’s summarization strengths. Intermediate results are persisted as augmented datasets and hosted via GitHub Pages, providing stable and reproducible data sources for subsequent visualization stages. The agent constrains generation to lightweight, interactive Vega–Lite specifications, integrates validation mechanisms to detect and repair errors, and produces a final gallery of interpretable, publication-quality visualizations. This design emphasizes flexibility across dataset types while ensuring efficiency, robustness, and narrative coherence in the resulting analyses.

1 INTRODUCTION

Large Language Models (LLMs) have rapidly emerged as powerful assistants for data analysis and visualization, demonstrating the ability to interpret unfamiliar datasets and produce structured outputs with minimal human input [1, 2]. Recent progress in LLM-based agents highlights their potential for automating visualization workflows across domains, lowering barriers for novice users and broadening access to advanced techniques. LLM-driven visualization is particularly attractive for rapid dataset exploration and report generation [3], yet producing reliable, readable, and publication-quality outputs remains challenging [4, 5]. Traditionally, visualization

design requires substantial expertise in data wrangling [6], encoding design decisions, and chart construction [7], which has motivated systems that recommend or constrain visualization design choices [8, 9]. The Agentic Visualization Challenge emphasizes automation, dataset-agnostic pipelines, and the generation of coherent, reproducible visualizations. To address these goals, we propose an agentic workflow for dataset-aware visualization generation that integrates structured planning, routing, and validation to ensure trustworthy results. For the Visualization Publication Dataset [10], our agent focuses on the *Abstract* column. This column is selected as it offers the richest semantic signal of research content. After selecting the column, our agent’s hybrid architecture combines deterministic text-mining with selective LLM usage: KeyBERT [11] extracts candidate terms, HDBSCAN [12] clusters them, and finally, the LLM generates concise, human-readable labels for the clusters. This design reduces redundant model usage while leveraging the summarization strengths of LLMs, yielding interpretable, semantically coherent categories that ground downstream visualizations in meaningful research themes.

We present a compact gallery of visualizations with descriptive captions, packaged in an interactive HTML report.

2 SYSTEM OVERVIEW

The agent executes a six-stage pipeline over an unfamiliar dataset in a CSV file. The pipeline is shown in Fig. 1. First, *Inspection and Routing* builds a schema preview, assigns a dataset type via a strict JSON contract with conservative fallbacks, and identifies the primary text field from which interpretable research areas can be derived. Next, *Analyzing Text* applies a hybrid pipeline combining KeyBERT, embeddings, and HDBSCAN clustering, followed by concise LLM-generated labels. *Plot Generation* produces compact, strict Vega–Lite only specifications to ensure portability and reproducibility. *Validation* then verifies field existence, encoding correctness, and overall readability. Although our pipeline encodes interactive elements, interactivity remains limited by the current capabilities of Vega–Lite. Finally, *Gallery Composition* deduplicates plots, adds captions, and assembles a self-contained HTML report. Our pipeline aligns with the challenge’s emphasis on automation, dataset-agnostic pipelines, and insightful outputs.

3 METHODS

This section details the technical implementation of each stage in the pipeline, including our agent’s state.

*e-mail: zaamir@cs.stonybrook.edu

†e-mail: boorboor@uic.edu

‡e-mail: ari@cs.stonybrook.edu

3.1 Inspection and Routing

The first stage of the workflow generates a dataset preview, consisting of the schema and a small sample of rows, and forwards it to a schema-aware router. The router classifies the dataset as one of `text_corpus`, `tabular_generic`, `time_series`, `graph_edges`, `geo`, with a conservative fallback to `tabular_generic` to ensure robustness when schema inference is uncertain. We selected these dataset types because they span a broad range of common dataset categories frequently encountered in visualization research and practice [13].

It returns a strict JSON contract specifying the dataset type, per-column roles, representative examples, and rationales with confidence scores. This contract guides subsequent stages, constrains the agent to Vega–Lite compatible transforms, and prevents invalid field references. In addition, the router ranks the top three candidates for the primary text field, the primary text field is defined as the column that best conveys the topic and overview of each record in the dataset. In the current dataset, it deterministically selects the *Abstract* column. An example contract is shown in Fig. 2.

```
{
  "dataset_type": "text_corpus",
  "columns": [
    "Abstract": {
      "role": "text",
      "example": "..."
    }
  ],
  "rationale": ".",
  "confidence": 0.82
}
```

Figure 2: Truncated JSON snippet showing per-column roles in the schema contract.

3.2 Analyzing Text

Once the primary text field is identified, we apply KeyBERT (`all-mnlp-base-v2`) [11] to extract high-relevance phrases, which are then normalized, embedded, and reduced with PCA before clustering with HDBSCAN [12]. Oversized clusters are recursively split to control growth and prevent excessive downstream LLM usage. The LLM is reserved for the final step, where it generates concise, human-readable labels that balance interpretability and efficiency. The resulting augmented dataset, which includes cluster assignments and labels, is published via GitHub Pages as a persistent resource for the agent.

3.3 Plot Generation

We employ a two-stage workflow to generate visualizations in a controlled and reproducible way. In the first stage, a *planner* agent is prompted to produce a structured JSON plan specifying the chart type, encoding channels, and a brief rationale for its design choice. This step is performed at a deterministic, low-temperature setting to minimize variance and ensure stable outputs across repeated calls. The planner’s output externalizes design intent while remaining lightweight, avoiding the overhead of full chart specifications at this stage.

In the second stage, a *coder* agent consumes the planner’s JSON and produces a complete Vega–Lite specification. Here, the agent enforces strict guardrails: the data source must be the augmented dataset, encodings must reference existing schema fields, and transforms are restricted to Vega–Lite operators. This design guards against hallucinated fields or unsupported constructs and ensures the generated visualizations are portable, scalable, and interpretable. By staging the process, we limit token usage, avoid redundant free-form generations, and reserve the LLM’s reasoning capacity for the parts of the pipeline where abstraction and design

decisions are being made, where it adds the most value. Example output is shown in Fig. 3.

```
# Example: high-level plan request
plan_req = {
  "fields": [
    {"name": "Year", "type": "temporal"},
    {"name": "Area", "type": "nominal"}
  ],
  "task": "Show how research clusters evolve over time."
}

# Planner returns structured JSON
{
  "chart_type": "area",
  "encoding": {"x": "Year", "y": "papers", "color": "Area"},
  "insight": "Tracks growth and decline of topics",
  "title": "Topic Evolution by Year"
}
```

Figure 3: Planner stage output: structured JSON specifying chart type, encodings, and rationale before translation into a full Vega–Lite specification.

The generated plan is then translated by the coder into a full Vega–Lite specification. This separation between planning and coding improves reproducibility, reduces token costs, and enhances explainability of the workflow by clearly documenting design intent before execution.

3.4 Validation and Refinement

In the final stage, interactive elements are encoded within each plot, after which the agent is supplied with the finalized Vega–Lite specifications. The agent then generates concise captions that highlight key insights, ensuring clarity and narrative coherence across visualizations. This process refines the outputs, validates their readability, and ties the story together.

3.5 Final Gallery

Our final gallery consists of three interactive plots, available [here](#).

4 LIMITATIONS

Through the two-step node plot generation process, the agent functions best on relatively basic functions which significantly caps interaction complexity. Additionally, our cluster quality depends on keyword extraction and density-based clustering choices, and labels inherit any bias present in the text. Although we curb token usage by deferring the LLM to the final stage, label quality can degrade for small or highly overlapping clusters.

5 CONCLUSION

We presented an agentic workflow for dataset-aware visualization that couples deterministic text mining with selective LLM usage, a planner-coder split for controlled visualization specification, and a validator to enforce schema compliance. By constraining visualization generation to Vega–Lite and hosting an augmented dataset online, our system favors portability, reproducibility, and cost efficiency, while still leveraging LLM strengths.

REFERENCES

- [1] Yuheng Zhao, Yixing Zhang, Yu Zhang, Xinyi Zhao, Junjie Wang, Zekai Shao, Cagatay Turkay, and Siming Chen. Leva: Using large language models to enhance visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 31(3):1830–1847, 2024. 1
- [2] Karim Huesmann and Lars Linsen. Large language models for transforming categorical data to interpretable feature vectors. *IEEE Transactions on Visualization and Computer Graphics*, 2024. 1

- [3] Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. Vi-seval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics*, 2024. 1
- [4] Alexander Bendeck and John Stasko. An empirical evaluation of the gpt-4 multimodal language model on visualization literacy tasks. *IEEE Transactions on Visualization and Computer Graphics*, 2024. 1
- [5] Yuan Cui, W Ge Lily, Yiren Ding, Lane Harrison, Fumeng Yang, and Matthew Kay. Promises and pitfalls: Using large language models to generate visualization items. *IEEE Transactions on Visualization and Computer Graphics*, 2024. 1
- [6] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, David Brodbeck, and Paolo Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011. 1
- [7] Tamara Munzner. *Visualization Analysis and Design*. CRC Press, 2014. 1
- [8] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016. 1
- [9] Dominik Moritz, Chenglong Wang, Gregory L. Nelson, Huy Nguyen, Adam M. Smith, Bill Howe, and Jeffrey Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):438–448, 2019. 1
- [10] Petra Isenberg, Florian Heimerl, Steffen Koch, Tobias Isenberg, Panpan Xu, Charles D. Stolper, Michael Sedlmair, Jian Chen, Torsten Möller, and John Stasko. vispubdata.org: A metadata collection about IEEE visualization (VIS) publications. *IEEE Transactions on Visualization and Computer Graphics*, 23(9):2199–2206, September 2017. 1
- [11] Maarten Grootendorst. Keybert: Minimal keyword extraction with bert embeddings. url`https://doi.org/10.5281/zenodo.4461265`, 2021. Software; version v0.1.3 archived on Zenodo. 1, 2
- [12] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017. 1, 2
- [13] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *The craft of information visualization*, pages 364–371. Elsevier, 2003. 2