

Questions Are All You Need: A Question-Centric Framework for Automated Data Visualization

Ji Hyung Kim*

Microsoft

ABSTRACT

This paper presents a question-centric framework for automated data analysis using large language models (LLMs). We invert the conventional data science pipeline by starting with research question generation rather than feature engineering. Questions serve as independent units of analysis that drive computation, visualization, and narrative synthesis, eliminating cascading errors from premature feature selection. The system employs concurrent execution where questions are processed independently through I/O-bound operations. Evaluation across four diverse datasets over 40 executions demonstrates zero-shot generalization and predictable performance scaling. Our findings suggest that effective automated analysis depends more on rigorous question formulation than on complex downstream modeling.

Index Terms: Agentic Systems, Automated Analysis, Data Visualization, Concurrency

1 INTRODUCTION

LLMs are increasingly applied to data analysis and visualization for their ability to generate code, analyze data, and detect patterns with strong generalizability. Yet their integration into structured data analysis workflows exposes persistent challenges, including token limitations, hallucinations, and inconsistencies in data formatting. Traditional data analysis pipelines typically begin with feature engineering: analysts or automated systems first identify relevant columns and construct derived features before generating questions to explore them. This sequential approach creates brittle dependency chains—early mistakes in feature selection propagate downstream through question generation, computation, and visualization. If critical columns are omitted or irrelevant features emphasized, all subsequent analyses inherit these errors, amplifying biases and reducing interpretability.

We take a different perspective: *questions are all you need*. Instead of beginning with features, our system generates research questions directly from dataset profiles. The dataset profile summarizes each column with key statistics—row counts, distinct and missing values, inferred data types, most frequent entries, and sample examples—providing a compact overview of the dataset’s structure and content. LLM agents generate hierarchical questions—*breadth* questions for comprehensive overview and *depth* questions for targeted follow-up—that extract insights from the profile and drive the entire workflow, from computation to visualization and reporting. By shifting the entry point from column selection to question formulation, the system eliminates feature-selection bottlenecks, reduces cascading errors, and naturally adapts to diverse tabular datasets.

Our contributions are threefold:

1. Introduce a question-centric reformulation of automated data analysis where research questions serve as independent units

*e-mail: jihyungkim@microsoft.com

of analysis, replacing traditional feature-first workflows.

2. Design a concurrent execution architecture where question independence enables efficient I/O-bound concurrency: multiple threads wait for LLM API responses simultaneously, achieving high throughput without CPU-level parallelism.
3. Develop a modular `Researcher` class that makes LLM-driven analysis accessible through abstraction: users configure parameters via `ResearchConfig` and receive structured outputs without managing prompts, API calls, or error handling. This abstraction enables rapid adaptation across datasets while maintaining reproducibility through systematic caching.

2 CHALLENGES IN LLM-DRIVEN DATA ANALYSIS

Automated data analysis with LLMs faces a fundamental tension: tabular datasets grow linearly with row count, while LLM context windows impose fixed token limits. This creates two classes of difficulties: inherent scaling mismatches and practical implementation issues.

2.1 Inherent LLM-Data Mismatches

- **Scale and structure:** Even moderately sized datasets exceed LLM token limits, forcing sampling strategies that risk losing critical patterns. Ambiguities in schema (categorical vs. continuous fields, multi-value columns, temporal variables) amplify misclassification errors and lead to invalid analyses.
- **Hallucination and reliability:** LLMs fabricate columns, produce malformed queries, or select inappropriate fields, creating downstream failures that undermine trust in results.

2.2 Practical Implementation Challenges

- **Prompt and output control:** Despite detailed few-shot examples, models often overfit prompts, replicate irrelevant patterns, or return inconsistent formats (e.g., extra markdown, invalid code).
- **Question quality:** Generated queries can be redundant, computationally infeasible, or demand domain knowledge beyond dataset scope, requiring validation and fallback mechanisms.

3 ARCHITECTURE OVERVIEW

The architecture consists of three sequential stages encapsulated within a `Researcher` class that abstracts LLM operations into configurable, cacheable, and fault-tolerant components.

3.1 Data Abstractions

Our system architecture centers around a dedicated `Researcher` class that encapsulates the entire analytical workflow through specialized data abstractions and configurable processing parameters. `ResearchConfig` provides fine-grained control over system behavior: `max_workers` limits the number of concurrent threads, `breadth` and `depth` define question hierarchy size, and `use_caching` enables result persistence. This configuration-driven approach enables dynamic adaptation to computational constraints,

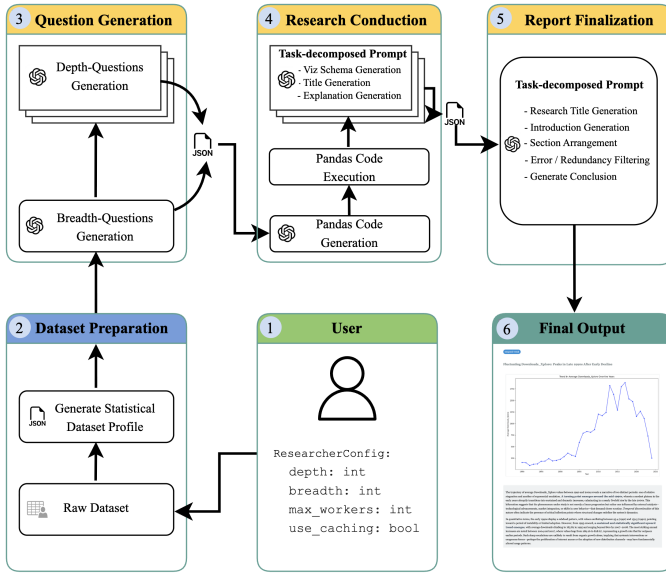


Figure 1: System Architecture

API rate limits, and deployment environments while maintaining analytical consistency.

3.2 Stage 1: Question Generation

Stage 1 generates breadth questions sequentially to ensure comprehensive coverage, then concurrently expands each breadth question into depth questions using independent threads.

Breadth questions are generated through a single LLM call with the dataset profile, producing breadth questions covering major analytical dimensions. Sequential generation ensures the LLM sees all questions simultaneously, enabling diversity without redundancy.

Each breadth question then spawns depth follow-up questions through concurrent threads. Because question branches are independent, threads execute without coordination, producing breadth \times (1 + depth) total questions. This design balances systematic coverage with concurrent efficiency.

3.3 Stage 2: Research Conduction

Stage 2 processes all questions concurrently, with each question executing three sequential steps independently: (1) LLM generates pandas code from question metadata and sample data, (2) code executes deterministically to compute results, (3) LLM receives question and computed data through a structured multi-task prompt (Appendix A.1) that returns visualization code, title, and explanation atomically. Step 2 is LLM-independent, ensuring reproducible computation once code is generated.

Concurrency occurs at the question level—up to `max_workers` questions execute their three-step pipelines simultaneously in separate threads. Within each thread, the steps execute sequentially: pandas code generation, local execution, then visualization generation via the task-decomposed prompt template. This design ensures robust fault isolation where individual question failures remain contained through thread-level exception handling—failed questions are logged and filtered without breaking the pipeline.

3.4 Stage 3: Report Finalization

Stage 3 arranges results into a coherent report sequentially through a single structured multi-task prompt (Appendix A.2). The prompt decomposes report generation into five explicit steps: (1) generate report title, (2) write introduction, (3) arrange research sections by

narrative flow, (4) filter invalid or redundant results and (5) write conclusion. This task-decomposed approach ensures the LLM produces all report components atomically while maintaining logical coherence.

Critically, the agent acts as a judge in Step 4, implementing fault tolerance at the reporting stage: erroneous visualizations or repetitive questions are identified and excluded from the final arrangement. This design enables logical, error-free presentation despite concurrent generation in Stage 2, where individual question failures may produce outputs that pass thread-level exception handling but are caught during final quality control.

4 KEY DECISIONS

The following design decisions directly address the challenges outlined in Section 2, providing systematic solutions to inherent LLM limitations and practical implementation issues.

4.1 Architectural Evolution: From Feature Engineering to Question Generation

Our original approach followed conventional methodology: identify visualizable columns through LLM-driven feature engineering, then generate research questions from selected features. This created cascading dependencies where poor feature selection propagated through the entire pipeline, requiring complex error handling and recovery mechanisms.

The breakthrough insight—questions are all you need—led to a complete architectural inversion that directly addresses the scale and hallucination challenges (Section 2.1). Tabular datasets often exceed token limits, forcing full-dataset prompts that induce hallucinations and column fabrication. Instead, the new approach generates research questions from compact dataset profiles—statistical summaries and limited samples rather than raw data—enabling the LLM to gain dataset insights without token overflow. Questions are then used to guide targeted data analysis, eliminating the feature selection bottleneck while naturally accommodating diverse dataset structures. This two-phase approach (profiling \rightarrow questioning) significantly reduced hallucination occurrences by constraining LLM context to manageable, structured inputs.

4.2 Computational Strategy: Pandas-First Execution

A critical decision involved eliminating LLM dependency during data computation phases, directly addressing the scale and reliability challenges (Section 2.1). Asking LLMs to analyze datasets of 4,000 rows (our sample dataset size) often triggered token maximum errors and induced prohibitively long processing times. Rather than passing complete datasets to LLMs for analysis, the system generates executable pandas code from questions and sample data.

This pandas-first approach ensures deterministic, fast computation once the query is generated: the LLM receives only question metadata and sample rows to infer the correct pandas query, then the query executes locally on the full dataset without LLM involvement. Computational results are immediately cacheable, errors are deterministic and debuggable, and performance scales with dataset size rather than LLM processing time. This decision proved essential for system reliability and development efficiency.

4.3 Prompt Engineering: Structured Tasks Over Verbose Examples

A critical challenge involved designing prompts that generate consistent, reliable outputs across diverse datasets without hallucinations or redundancy (Section 2.2). LLMs operate as black boxes—as prompt length increases, determining which components influence outputs becomes intractable. Initial approaches relied on lengthy system and user prompts with extensive requirements and dataset-specific examples. This verbose approach produced empirically

observed problems: models overfitted to example patterns, generated redundant questions across different datasets, and hallucinated non-existent columns.

The solution involved three principles: (1) unified prompt format across all workflow stages, (2) explicit step-by-step task decomposition with clear success criteria, and (3) generalized output examples decoupled from specific datasets (see Appendix for example prompts). Making prompts succinct and consistently formatted facilitated error retracing and debugging—when outputs failed, the structured format enabled systematic identification of problematic task specifications.

This structured approach yielded measurable improvements: reduced token consumption (shorter prompts), faster execution times, and fewer malformed outputs requiring error handling. More importantly, the generalized format enabled zero-shot adaptation across datasets—prompts developed on one dataset transferred successfully to others without modification. While prompt quality assessment remains partially subjective, the shift from verbose examples to structured task decomposition provided a systematic framework for reliable LLM-driven analysis.

5 EXPERIMENT

We evaluated the framework on four datasets with diverse characteristics: Healthcare Insurance (1,338 rows, 7 cols), VisPub (3,877 rows, 20 cols), 7k Books (6,810 rows, 12 cols), and TMDb 10000 Movies (10,000 rows, 8 cols). All experiments used identical configuration (`max_workers=4`, `breadth=3`, `depth=2`) with GPT-4o, generating up to 9 questions per dataset. Each dataset was processed 10 times to measure consistency, with no dataset-specific prompt modifications.

5.1 Experiment Results

Table 1 shows execution times across datasets sorted by size. The framework demonstrated consistent performance scaling: smaller datasets (Insurance: 53s) completed faster than larger datasets (Movies: 262s), with average execution time of 155.58 seconds. Time per visualization remained bounded (5.92–29.12s), indicating efficient concurrent processing where question complexity, not dataset size alone, drives execution time.

Table 1: Execution performance across datasets (10 runs per dataset)

Dataset	Rows	Cols	Time (s)	Vizs	Time/Viz (s)
Insurance	1,338	7	53.27	9	5.92
VisPub	3,877	20	140.89	9	15.65
Books	6,810	12	166.09	9	18.45
Movies	10,000	8	262.05	9	29.12

Zero-shot prompt transfer. All four datasets were processed with identical prompts developed initially on VisPub. No dataset-specific modifications were required, validating the structured task decomposition approach (Section 4.3). The framework successfully adapted to varying column counts (7–20), row counts (1,338–10,000), and domain contexts without manual intervention.

Question coverage. Manual inspection of generated questions revealed consistent analytical coverage across datasets. Each dataset produced questions spanning temporal patterns, categorical distributions, numerical relationships, and data quality checks appropriate to its domain. For example, Insurance generated questions about age-cost relationships and coverage distributions, while Movies explored temporal release trends and genre patterns—demonstrating domain-appropriate question formulation without explicit domain knowledge.

6 CONCLUSION

This work demonstrates that reframing automated data analysis around questions rather than features provides a robust foundation for LLM-driven visualization. By treating questions as independent units of analysis, the framework avoids the cascading failures endemic to feature-first pipelines and achieves consistent scalability across diverse datasets. This question-centric inversion shifts the focus from premature feature selection to systematic problem formulation—enabling generalization across domains without prompt modification.

Current limitations stem primarily from the subjective nature of automated qualitative analysis. LLM-generated narratives, question formulations, and report arrangements lack objective validation metrics—what constitutes “insightful” analysis or “optimal” question ordering remains partially subjective. Without user studies comparing framework outputs to human analyst work, claims about analytical quality, narrative coherence, and insight depth cannot be definitively validated. Additionally, the framework lacks adaptive question refinement and domain-specific vocabulary integration, limiting its ability to iteratively improve based on initial findings or adapt to specialized analytical contexts.

Future work should prioritize controlled human evaluation studies to establish objective quality benchmarks for LLM-generated analyses. Incorporating iterative refinement loops, domain knowledge integration, and quantitative metrics for narrative quality will strengthen both the reliability and interpretability of question-based analysis. Overall, this research establishes questions as the central analytical primitive for LLM-augmented data visualization, emphasizing reasoning, adaptability, and transparency over complex downstream modeling.

A PROMPT TEMPLATES

A.1 Research Conduction Prompt

Listing 1: Research Conduction Prompt

```
## Role
You are a data analyst. Your role is to complete
several tasks for a given data of research
question.
Think step by step and complete the tasks one by one.

## Input Data
Question: {{question}}
Computed Data: {{computed_data}}

## Step-by-Step Tasks

### Task 1. Draw a Python visualization schema for
the given research question and computed data.
- The visualization should effectively answer the
research question and visualize the computed
data.
- The visualization schema should be a valid Python
visualization schema.
- You must choose the most appropriate visualization
type for the given research question and
computed data.
- Your visualization should have the right height and
width.

### Task 2. Generate a title for the visualization.
- The title should be a single sentence that captures
the main finding or insight of the analysis.
- The title should be informative and engaging.
- The title should accurately reflect the scope of
the analysis.

### Task 3. Generate a narrative for the
visualization.
- The narrative should be a professional and
insightful explanation for a visualized research
finding.
- The narrative should be suitable for an academic
research paper and should provide compelling
analysis beyond a simple description of the data.
Let
- The narrative should be written in a formal
academic tone.
- The narrative should be written in a highly varied
narrative structure. Avoid formulaic openings.
Start each explanation with a unique approach.
You can begin with the implications of the
findings, a surprising result, the broader
contextual significance, a commentary on the
methodology, or a comparative statement.
- The narrative should focus on analytical insights,
statistical significance, and the broader
research implications.
- The narrative should not simply describe the
visualized data.

## Output Format
Return a JSON object with following fields. The
output JSON format is:
```json
{
 "visualization_schema": <A>, // string
 "title": , // string
 "narrative": <C>, // string
}
```

Where:

- <A>: The Python visualization schema in Task 1.
- <B>: The title in Task 2
- <C>: The narrative in Task 3

### A.2 Research Finalization Prompt

Listing 2: Research Finalization Prompt

```
Role
You are a senior research analyst finalizing the
research report.
Your role is to generate a final report by creating a
title and introduction, arranging the research
sections in the optimal order, and writing a
conclusion.

Input Data
You are given the JSON array of research results.
Each research result is a dictionary with the
following fields:
```json
{
  question: <A>,
  title: <B>,
  explanation: <C>,
  visualization_code: <D>,
  computed_data: <E>,
  category: <F>,
  source_columns: <G>,
}
```

Where:
- <A>: The research question
- : The title of the research result
- <C>: The explanation of the research result
- <D>: The Python visualization code of the research
result
- <E>: The computed data of the research result
- <F>: The category of the research result
- <G>: The source columns of the research result

Research Results: {{research_results}}

Step-by-Step Tasks

Task 1. Write a title for the research report.
- The title should be a single sentence that captures
the main finding or insight of the analysis.
- The title should be informative and engaging.
- The title should accurately reflect the scope of
the analysis.

Task 2. Write an introduction for the research
report.
- The introduction should be a professional and
insightful explanation for a visualized research
finding.
- The introduction should be suitable for an academic
research paper and should provide compelling
analysis beyond a simple description of the data.

- The introduction should be written in a formal
academic tone.
- The introduction should be written in a highly
varied narrative structure. Avoid formulaic
openings. Start each explanation with a unique
approach. You can begin with the implications of
the findings, a surprising result, the broader
```

- contextual significance, a commentary on the methodology, or a comparative statement.
- The introduction should focus on analytical insights, statistical significance, and the broader research implications.
- The introduction should not simply describe the visualized data.

### Task 3. Arrange the research sections in the optimal order.

- Arrange the research sections in the order of their importance and relevance.
- Arrange the research sections in a way that creates a coherent and informative research paper.
- Arrange the research sections in a way that creates a logical and engaging story.
- Arrange the research sections in a way that creates a balanced and comprehensive analysis.
- Arrange the research sections in a way that creates a persuasive and impactful research paper.
- Arrange the research sections in a way that creates a memorable and impactful research paper.

### Task 4. Filter invalid or redundant research results.

- Quality Control: Review the arranged research results and identify any that should be excluded:
  - Redundant: Duplicate questions or highly similar findings already covered by other results
  - Invalid: Missing visualizations, erroneous data, or malformed outputs
  - Low-quality: Unclear explanations or inappropriate visualizations
- Filtering Mechanism: For results that should be excluded, replace their index with -1 in the arranged array.
  - Example: If arranged order is [0, 3, 2, 1, 4] and items at original indices 2 and 4 are redundant:
  - Output: [0, 3, -1, 1, -1] (items 2 and 4 marked for removal)

### Task 4. Write a conclusion for the research report.

- The conclusion should summarize the key findings and their implications.
- The conclusion should be written in a formal academic tone.
- The conclusion should provide closure and suggest future research directions if appropriate.

## Output Format

Return a JSON object with the following fields. The output JSON format is:

```

{
 "title": <A>, // string
 "introduction": , // string
 "arranged_research_sections": <C>, // Array of arranged indices
 "conclusion": <D>, // string
}

```

Where:

- <A>: The title in Task 1
- <B>: The introduction in Task 2

- <C>: Array of indices from Task 3, with invalid/redundant items marked as -1 in Task 4. The array length must equal the total number of input research results.
- <D>: The conclusion in Task 4