Are LLMs ready to help non-expert users to make charts of official statistics data?

Anonymous Author(s)

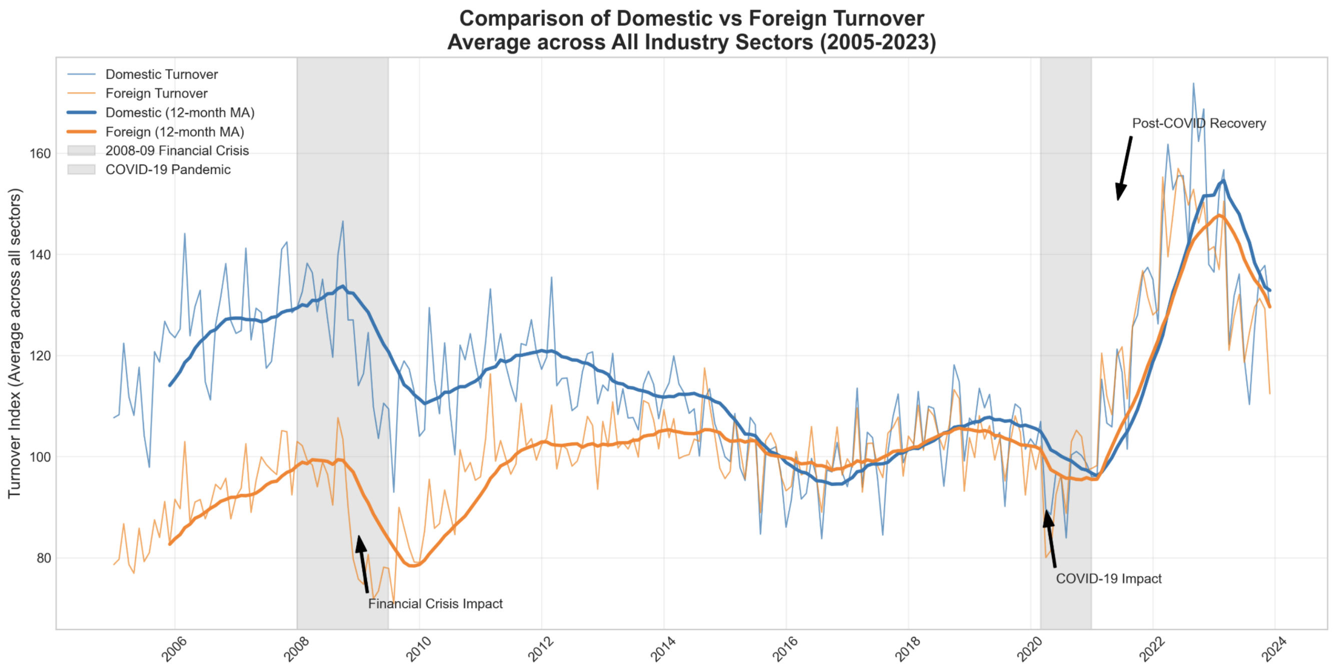


Figure 1: Sample visualization generated by Claude 3.7 using a modular prompt and a self-feedback loop (7 iterations), responding to the prompt: *"Plot and compare foreign and domestic turnover for All Sectors."* The chart compares domestic and foreign turnover indices from 2005 to 2023, averaged across all industry sectors, based on data from Statistics Netherlands. Raw monthly trends are shown with thin lines, while 12-month moving averages are overlaid with thicker curves to highlight longer-term patterns. Two key economic disruptions (the 2008–2009 Financial Crisis and the COVID-19 Pandemic) are marked with shaded regions. The visualization showcases use of reasoning, clear axis labeling, and appropriate use of smoothing and context. Sample figures are displayed as produced by the LLM system, without any post-processing. More results can be found at https://anonymous.4open.science/r/llm_visualization_results-2F67/

Abstract

In this time when biased information, deep fakes, and propaganda proliferate, the accessibility of reliable data sources is more important than ever. National statistical institutes provide carefully curated datasets that contain quantitative information on a wide range of topics. However, that information is typically spread across many tables and just the plain numbers may be arduous to process. Hence, even though the information is openly accessible, it may be practically inaccessible and unintelligible. We ask the question *"To what degree are current Generative AI models capable of facilitating the identification of the right data and the fully-automatic creation of charts to provide information in visual form, corresponding to user queries about data from Statistics Netherlands?"*. We present a structured evaluation of recent large language models' (LLMs) capabilities to generate charts from complex data in response to user queries. Working with diverse public tabular data from Statistics Netherlands, we assessed multiple LLMs on their ability to identify relevant data tables, perform necessary manipulations, and generate appropriate visualizations autonomously. We propose a thorough new evaluation framework spanning three dimensions: data retrieval & pre-processing, code quality, and visual representation. Results indicate that locating and processing the correct data represents the most significant challenge. Additionally, LLMs rarely implement visualization best practices without explicit guidance. When supplemented with information about effective chart design, models showed marked improvement in representation scores. Furthermore, an agentic approach with iterative self-evaluation led to excellent performance across all evaluation dimensions. These findings suggest that LLMs' effectiveness for automated chart generation can be substantially enhanced through appropriate scaffolding and feedback mechanisms, and that systems can already reach the necessary accuracy across the three evaluation dimensions.

Keywords

Data Visualization, Official Statistics, Generative AI, Text-to-vis

1 Introduction

Official statistics data are more relevant than ever, but are inaccessible. In an era increasingly characterized by the proliferation of misinformation, ensuring public access to and understanding of reliable official statistics is becoming more and more important [12]. National statistical institutes, such as Statistics Netherlands (CBS), serve as official providers of such authoritative datasets, which are vital for informed public discourse and data-driven decision-making by governments and citizens alike. However, these official statistics datasets are often large-scale, complex, and laden with domain-specific terminology, posing significant barriers to direct interpretation, especially for non-expert users.

On-demand retrieval and chart generation using GenAI may improve accessibility. Data visualization is a powerful means to distill insights from complex information, yet creating effective charts traditionally demands considerable technical expertise and time. This limits the ability of many individuals to independently explore and understand official data. The recent rise in capabilities of Large Language Models (LLMs) presents an opportunity: their capacity to translate natural language requests into executable code, including visualization scripts, promises to democratize data analysis. By enabling users to interact with the official data in plain language, LLMs could significantly lower the thresholds to data-driven decision-making and improve data literacy overall.

GenAI systems may face several challenges and it is largely unknown how well they may tackle these. Despite this promise, the true readiness of LLMs to assist non-expert users in generating meaningful and accurate charts from the highly structured, real-world data typical of official statistics remains an open question. Such a task is non-trivial; an LLM must not only (i) correctly interpret a user's query, potentially expressed using everyday language rather than precise domain jargon, but also (ii) identify the correct data table from potentially thousands, (iii) perform necessary data manipulations like filtering and aggregation accurately, (iv) select an appropriate chart type for the data and query, and (v) generate syntactically correct code that adheres to visualization best practices. An error at any stage can lead to flawed or misleading visualizations, undermining the goal of clear communication. If these challenges can be overcome, LLMs could enable a broad audience, from researchers and journalists to the general public, to obtain reliable visuals from authoritative data sources in seconds, thereby widening access to official statistics.

Contributions. To explore the current possibilities, we designed 25 questions of various difficulty that may be answered using data from seven frequently-accessed tables from Statistics Netherlands, and tested the capabilities of eight state-of-the-art LLMs of varying sizes to generate corresponding charts that would answer these questions. We created a comprehensive evaluation framework to score each attempt by the models on 22 unambiguous binary questions—covering visual clarity, data correctness, and code soundness—yielding quantitative, model-agnostic metrics. These questions are specifically selected to not only base the evaluation solely on the generated design but also the quality of code and relevance. Through this study, we aim to answer the question to what degree is it feasible (with current tools) to incorporate said models in a data visualization pipeline for a real-world use case.

In summary, this research delivers three contributions: (1) an agentic system that combines targeted prompt engineering with iterative self-correction for natural-language chart generation on official statistics; (2) a comprehensive evaluation framework for assessing the model performance; and (3) a comparative study showing how model size, provided context, and task difficulty each influence end-to-end visualization quality.

The evaluation results suggest that, when supplied with a good scaffold such as compact design guidance and the freedom to debug their own outputs, modern LLMs can already produce highly effective charts for real-world queries, opening a path toward accessible, language-driven analytics for non-experts.

Outline. We first review related work on automated data visualization and LLM applications in Section 2. In Section 3 we present the system design, including prompt design and the implementation of both zero-shot and agentic LLM systems. In Section 4, we detail the structured evaluation framework with criteria across visual, data, and code dimensions. Results are presented in Section 5, and we discuss key findings and conclusions in Section 6, with implications for future research and real-world deployment.

2 Related Work

Recent advances in large language models (LLMs) have expanded their applications into areas traditionally requiring substantial domain expertise, including data visualization. Conventional tools such as D3.js, Vega-Lite, Plotly Express, and Tableau provide powerful capabilities for visual representation but effective use of these tools often demands familiarity with programming concepts, chart design principles, and domain-specific data structures.

In response, research interest in automating the visualization process has grown rapidly, aiming to lower the technical barriers to data exploration and communication. The Machine Learning for Visualization survey [20] highlights a notable surge in visualization-related studies, particularly since the widespread adoption of LLMs. The potential to translate natural language queries directly into high-quality visual outputs offers an exciting opportunity to make data analysis accessible to a broader audience. Yet, De Bie et al. [1] argue that, in the context of data science, the parts of the workflow outside of pure model construction, such as data preprocessing, exploratory analysis, results communication remain especially hard to automate due to their open-ended, domain-specific nature that is reliant on human judgement. So, achieving reliable and precise visualization generation remains a significant technical challenge.

To address these challenges, a variety of toolkits and systems have been developed, leveraging natural language processing and machine learning techniques. NL4DV converts English queries into JSON analytic specifications and corresponding Vega-Lite charts [11]. ChartGPT extends this idea by fine-tuning FLAN-T5-XL on a six-stage reasoning pipeline, tackling vague or incomplete requests through step-wise code generation [16]. Recommendation engines such as DeepEye [13] and VizML [6] predict suitable chart encodings from data characteristics, while LLM4Vis employs few-shot prompting with ChatGPT to suggest multiple alternatives together with textual justifications [19]. Integrated frameworks then began to appear: LIDA orchestrates four modules, namely summarizer, goal explorer, visualization generator, and infographer, around GPT-3.5 to move from raw table to full infographic in a single workflow [4]. At the fully automated end, Data2Vis maps tabular inputs to Vega-Lite specs with a sequence-to-sequence recurrent network, eliminating the need for handcrafted templates [5].

The arrival of stronger foundation models has encouraged lighter, prompt-centric designs. CHAT2VIS demonstrates that carefully engineered prompts alone can push GPT-3/Codex to produce runnable matplotlib scripts and even handle multilingual or ambiguous utterances [10]. Prompt4Vis further boosts zero-shot performance by automatically mining in-context examples and pruning irrelevant schema tokens before each LLM call, reducing the gap with bespoke neural architectures [9]. MatPlotAgent is an agentic framework that writes MatPlotLib code and was evaluated for scientific data visualization [21]. Finally, VisPath refines the agentic approach of MatPlotAgent to enhance performance, as evaluated on two code-focused benchmarks [15]. Although these studies bear similarity to ours, we rely fully on manual evaluation of the results using a comprehensive set of binary evaluation questions and our prompts are much less specific and more in line with use by non-experts.

Complementing these heavyweight engines, retrieval-augmented solutions such as VIST5 combine a compact, fine-tuned T5-base model with an external memory of few-shot examples, yielding domain-portable agents that run in real time on commodity hardware [18]. In parallel, Visistant combines Google's Gemini-Pro with LangChain conversational memory, illustrating how modern chat frameworks can support iterative refinement—users can zoom, rescale, or alter encodings with follow-up instructions [14]. Together, these studies chart a spectrum of design trade-offs between accuracy, latency, and resource footprint.

Nevertheless, despite the promising trajectory of these systems, critical challenges remain regarding the accuracy, reliability, and adaptability of LLM-driven visualization generation. Recent evaluations [17] reveal that although models such as GPT-4 achieve considerable success (it can render roughly 80% of proposed charts) in generating visualizations using libraries like matplotlib, Seaborn, and Plotly, they are still prone to a range of errors. These shortcomings become particularly evident when dealing with complex datasets or ambiguous user queries, where context understanding and precise mapping are required. Furthermore, current evaluation frameworks are often limited in scope. Automated tools like VisEval [2] and VizLinter [3] have made progress in structuring evaluation around dimensions such as validity, legality, and readability. However, they typically assess outputs based on syntactic correctness rather than semantic fidelity or practical use, overlooking deeper aspects like effective communication, insightfulness, and task alignment.

A key limitation in the existing body of research is its focus on general-purpose datasets, often curated for academic or experimental purposes. The application of LLMs to official, structured data remains a relatively unexplored frontier. Official statistical datasets, such as those maintained by national statistical organizations, present particular challenges due to their complexity, size, and domain-specific characteristics. These datasets frequently involve hierarchical structures, specialized terminologies, and metadata that must be carefully interpreted for effective visualization.

Recent efforts at the Statistics Netherlands have highlighted these challenges. Kouwenhoven et al. [7] propose a pipeline for statistical question answering that leverages knowledge graphs and constrained decoding techniques to improve retrieval accuracy over large, structured datasets. Their approach significantly enhances query relevance but focuses mainly on text-based answers, without extending into the domain of visualization. In their research, Lageweg et al. [8] utilize schema-constrained decoding to ensure LLM outputs faithful to the data source, improving precision and relevance in statistical contexts. Although both systems demonstrate effective strategies for structured data retrieval, they stop short of integrating automatic visualization as a continuation of the user query process, leaving an important gap in the practical usability of these systems for exploratory data analysis.

In light of these developments, there is a growing recognition that simply generating static visualizations is insufficient. True automation requires systems that can retrieve relevant structured data, interpret complex queries, reason about suitable representations, and produce effective visual outputs within a unified framework. Building on earlier work in knowledge-constrained retrieval and iterative refinement, this paper addresses this need by proposing an agentic framework that enables language model agents to autonomously retrieve, process, and visualize official datasets based on natural language prompts. Through iterative feedback loops, modular prompt engineering, and structured evaluation, the framework aims to overcome limitations identified in existing systems.

3 System Design

This section introduces the pipeline that we design to evaluate LLMs capabilities for the visualization tasks. We describe the dataset retrieval process, the design of both zero-shot and agentic systems, the structure of prompts and modular instructions, and the architecture of the tool-equipped agent system

3.1 CBS Data

This study uses official statistics published by Statistics Netherlands (CBS), which provides structured datasets on a wide range of topics such as economy, demography, and industry. These datasets are made available through the CBS OData3 API, which also offers English-language access to metadata and raw tabular data in a machine-readable JSON format. Each dataset is organized into a table with a unique identifier, and includes structured metadata describing variables, dimensions, and data types. For instance, the table `85332ENG`, which contains data on births in Caribbean Netherlands, can be accessed using the following OData query: https://opendata.cbs.nl/ODataApi/OData/85332ENG/TypedDataSet.

3.2 Dataset Retrieval via Sentence Transformers

For the zero-shot approaches, the first step in the visualization pipeline involves identifying the most relevant dataset based on the user's natural language prompt. We leverage metadata made available through the OData3 API, which provides descriptions of thousands of structured data tables. These descriptions serve as the reference text corpus for retrieval.

To match user prompts with appropriate data tables, we employ Sentence Transformers—a family of Transformer-based models capable of generating semantically meaningful embeddings for sentences. After testing multiple candidate models (e.g., `paraphrase-MiniLM-L3-v2`, `multi-qa-MiniLM-L6-cos-v1`), we selected `all-mpnet-base-v2` due to its superior performance in correctly handling ambiguous or domain-specific terms. For example, it reliably distinguishes between "labour" in the context of employment statistics versus childbirth in fertility data, where smaller models failed.

The retrieval process involves: (1) pre-encoding each metadata description into a dense vector using the selected model and storing them for efficiency, (2) encoding user prompts at runtime with the same model, and (3) computing cosine similarity between the prompt and all stored embeddings. The dataset with the highest similarity score is selected as the most relevant match (top-1 policy; no human filtering). This embedding-based approach enables robust semantic matching beyond keyword overlap, effectively capturing synonyms and contextual meaning.

To assess retrieval effectiveness, we performed an exact match analysis across the 25 evaluation questions. The correct dataset appeared in the top 10 results for 80% of questions, in the top 5 for 68%, and ranked first in 36%. While this indicates that there is room for improvement, it also reflects the challenge of a highly overlapping corpus where many datasets may appear or be equally relevant. We found that most top-ranked results are at least good alternatives, containing either the requested or similar data. In all experiments below, we use the top-1 dataset returned by the retriever. A detailed breakdown of retrieval performance can be found at https://anonymous.4open.science/r/llm_visualization_results-2F67/.

3.3 Study Design and Evolution

Using the top-1 retrieved dataset (no manual filtering), our research followed a two-phase approach to comprehensively evaluate LLM capabilities for visualization tasks:

Phase 1: Zero-Shot Evaluation. We first conducted extensive experiments where LLMs were provided with dataset context and instructions, requiring them to generate complete Python visualization solutions in a single response without iteration or feedback.

Phase 2: Agentic System Development. Building on insights from Phase 1, we designed and implemented an agentic system that enabled LLMs to interact with data and code execution environments, enabling an iterative visualization development process.

This natural progression in our research methodology mirrors the evolution of LLM capabilities themselves, from constrained text generation to more sophisticated reasoning and agentic behaviors capable of solving complex tasks through iteration and feedback.

3.4 System Prompts and Input Engineering

Prompt design was a critical component in the research methodology, with distinct approaches for zero-shot and agentic systems:

Zero-Shot Prompt Template:
Data Analysis Request
Column names (attributes) of data to be analyzed:
[Column names listed here]

Sample row from the data (example):
[Sample data row values listed here]

Data description:
[Detailed dataset description]

[Optional instructional modules inserted here]

Additional information: Don't use the sample row as input. Make sure to use only the corresponding column name(s) from the list provided above. Assume that you already have access to all the data stored in a variable named df. Don't use any variables other than df and those derived from df. Now do the following: Write Python code to [specific visualization task]. Provide a short description of the data, separated from the code.

Figure 2: The zero-shot system prompt template used in our experiments. This template was populated with dataset-specific information and task requirements for each visualization request.

(1) **Zero-Shot System Prompt Design**: For our initial experiments, we designed a structured prompt template (Figure 2) focused on maximizing the LLM's ability to generate complete visualization code in one pass. The prompt structure evolved through several iterations, each addressing the limitations observed in earlier outputs (based on early example prompts, not using the 25 final evaluation prompts). Early versions of the prompt provided all of raw data directly to the model, which proved ineffective due to confusion about data types and subset selection. So, raw data was removed, and key components such as column names, sample row, and metadata descriptions were added to clarify the dataset structure without overwhelming the models. Additional clarifications regarding hardcoding and assumptions about the used `df` variable were added to ensure the model used the full dataset, guiding it to generate correct and generalizable code. As a result, the final prompt structure provided comprehensive context about the dataset while establishing clear constraints around code generation.

(2) **Agentic System Prompt Design**: For the agentic system, we developed a more sophisticated prompt (Figure 4) that guided the LLM through an interactive workflow. This prompt introduced several advanced elements:
- Specific tool definitions for environment interaction
- Structured workflow guidance through a seven-step process
- Explicit file paths and operational parameters
- Error handling and debugging instructions
- Verification and feedback mechanisms

As shown in the figures, both prompts were carefully structured to provide appropriate guidance while testing different capabilities. The zero-shot prompt emphasized comprehensive data context and clear coding constraints, while the agentic prompt focused on providing a structured workflow and tool interaction guidelines.

3.5 Modular Prompt Components

To systematically evaluate the impact of different instructional approaches, we implemented a modular prompt system with components that could be enabled or disabled:

```
Algorithm 1: Agent Runner (pseudocode)
Input: user prompt, retriever, tools (list_files, read_file_head, execute_python_code, read_visualization_image), max_iters
1: Retrieve top-1 dataset metadata and path
2: For t = 1..max_iters: plan → generate code → execute → read outputs/errors/image
3: If plot valid and matches request: break; else revise based on feedback
4: Save final code, plot, and logs
```

Figure 3: Pseudocode of the agentic loop.

Agentic Prompt Template:
You are an expert data analysis and visualization assistant. Your goal is to help the user create a visualization based on their request.
[Dataset context injected here]
[Enabled modules injected here]

You have access to a filesystem restricted to the './data/' directory. All outputs for this run will be saved within './output/[run_id]/'.
Key file paths for this run:
- Log File: [log file path]
- Executed Code: [code save path pattern]
- Target Plot: [target output path]
- Human Feedback: [feedback file path]

Follow these steps:
1. **Understand Request:** Clarify the user's goal.
2. **Explore Data:** Use `list_files` if needed.
3. **Inspect Data:** Use `read_file_head`.
4. **Plan Code:** Plan pandas/matplotlib/seaborn code.
5. **Execute Code:** Use `execute_python_code`.
6. **Analyze Results:** Check execution output.
7. **Respond:** Explain steps, results, describe plot.

Available Tools:
- `list_files`: Lists files in directory
- `read_file_head`: Reads start of file
- `execute_python_code`: Executes Python code
- `read_visualization_image`: Reads the plot
- `get_human_feedback`: Logs request for help

Figure 4: The agentic system prompt template used in our tool-equipped experiments. This template guided the LLM through a structured workflow while providing access to specialized tools for data exploration, code execution, and visualization verification.

(1) **Data Visualization Context**: Comprehensive principles for effective data visualization, covering core visualization principles, visual design rules, quality criteria, key considerations, and implementation guidelines.
(2) **Lessons Learned**: Domain-specific advice derived from previous visualization experiments, highlighting common errors in visual design, code implementation, and data representation to help the LLM avoid typical pitfalls.
(3) **Visualization Checklist**: A self-assessment framework prompting the LLM to evaluate its output against criteria such as clarity, data accuracy, and task alignment.

These modules were implemented as optional text blocks that could be injected into the system prompts for both zero-shot and agentic experiments, allowing us to measure their impact on visualization quality and task success rates across different LLM providers and difficulty levels.

3.6 Agentic System Architecture

The implemented agentic visualization system consists of three main components:

(1) **Agent Runner**: Acts as the central orchestrator of the workflow. The runner manages the conversation loop with the LLM, invokes tools based on the model's requests, establishes logging mechanisms, and maintains run-specific output directories for artifacts and performance tracking.
(2) **LLM Interface**: Facilitates communication with the underlying API. This component handles sending conversation history, system prompts, and available tool specifications to the LLM, and processes its responses into structured formats (text responses, tool calls, and reasoning traces).
(3) **Tool Suite**: Provides the LLM with a set of specialized functions that enable interaction with the environment. These tools allow for file system navigation, data inspection, code execution, visualization review, and human feedback solicitation when necessary.

File access is restricted to predefined directories (`data/` for input, `output/` for generated files) to ensure experimental control and reproducibility.

3.7 Agent Workflow

The agentic system's workflow, depicted in Figure 5, follows a structured sequence that enables iterative visualization development:

(1) **Request Understanding**: Comprehend and clarify the user's visualization goal
(2) **Data Exploration**: Navigate the filesystem to locate relevant datasets
(3) **Data Inspection**: Examine data structure and content to inform visualization planning
(4) **Code Planning**: Develop a strategy for implementing the visualization using Python libraries
(5) **Code Execution**: Generate and execute Python code with explicit file paths and output locations
(6) **Result Analysis**: Evaluate execution outcomes, debug errors, and verify visualization output
(7) **Response Generation**: Explain process, describe results, or request human intervention when necessary

3.8 LLM-Agent Capabilities

The LLM-based agent is equipped with five primary capabilities through its tool interface:

(1) **Data Discovery**: Using the `list_files` tool, the agent can navigate through the available datasets in the data directory to identify relevant files for a given visualization request.
(2) **Data Inspection**: Through the `read_file_head` tool, the agent can examine the first several lines or rows of a dataset to understand its structure, variables, and content before attempting visualization.
(3) **Code Generation and Execution**: The `execute_python_code` tool enables the agent to generate and run Python code using standard data science libraries (pandas, matplotlib, seaborn, numpy) to produce visualizations. The

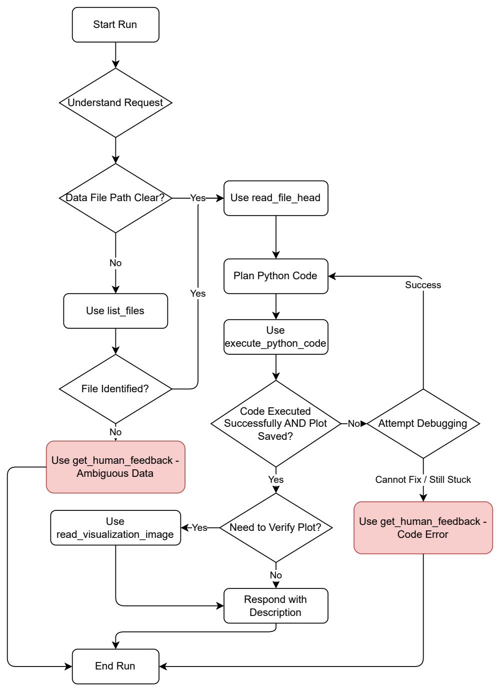


Figure 5: Typical decision flow of the agentic visualization system.

system automatically preserves each code iteration for analysis.

(4) **Visualization Verification**: Using `read_visualization_image`, the agent can inspect the generated visualization, verify its correctness, and potentially refine it based on the observed output.

(5) **Assistance Seeking**: When faced with unresolvable errors or ambiguities, the agent can use the `get_human_feedback` tool to document its reasoning process and request human intervention. This tool was never invoked in practice.

3.9 Design Rationale

We briefly justify key design decisions. After testing smaller sentence transformers, we decided to use `all-mpnet-base-v2` as it handled domain terms (e.g., labour) more robustly than lighter models, and outperformed sparse BM25 on semantic matches under short prompts. We study both a zero-shot setting (one-pass code) and an agentic setting to quantify the incremental value of iterative execution and self-correction. The tool suite is deliberately minimal (file listing, head inspection, code execution, image read) to mirror a realistic sandbox and reduce degrees of freedom.

4 Evaluation Methodology

4.1 Experimental Protocol

Our experimental evaluation followed a structured protocol to systematically assess the agent's visualization capabilities:

(1) **Task Selection**: We compiled a diverse set of visualization tasks based on official statistics datasets from Statistics Netherlands, varying in complexity from simple time series plots to multi-variable comparative visualizations. Tasks were categorized into three difficulty levels:
- **Easy**: Simple single-variable visualizations (e.g., "Plot the volume of cheese production in the Netherlands")
- **Medium**: Tasks requiring data filtering or multiple variables (e.g., "Plot the monthly volume of raw cow's milk delivered by dairy farmers between 2010-2015")
- **Hard**: Complex tasks with specific analytical requirements (e.g., "Plot the seasonally adjusted daily turnover for domestic and foreign markets for sector '16 Manufacture of wood products' between 2020-2021")

(2) **Dataset Preparation**: We utilized seven real-world official statistics datasets covering diverse domains:
- Milk supply and dairy production by factories
- Caribbean Netherlands births, fertility, and age of mother
- Consumer price indices
- Industry production and sales statistics
- Producer price indices
- Municipal accounts balance sheets
- Population demographics by sex, age, generation, and migration background

These datasets were selected to ensure broad coverage of both categorical and numerical data types, enabling the generation of a wide variety of visualization styles such as bar charts, line graphs, pie charts, heat maps, etc. The selection process also considered the structural complexity and ambiguity of attributes to evaluate how well LLMs manage real-world challenges. In addition to technical variability, usage metrics provided by Statistics Netherlands, such as visitor counts, session frequency, and returning user ratios, were analyzed to prioritize datasets frequently accessed by the public. This ensured that the evaluation and prompt refinement processes remained closely aligned with actual user behavior and practical relevance.

(3) **Experimental Approaches**: We implemented two distinct approaches to evaluate and compare LLM capabilities:
- **Zero-Shot Generation**: Tasks were presented to LLMs with dataset context and visualization instructions, requiring the model to generate a complete Python visualization solution in a single response without iteration or feedback.
- **Agentic Workflow**: The same tasks were presented to LLMs equipped with tools for data exploration, code execution, visualization verification, and error recovery, enabling an iterative development process similar to human data scientists.

(4) **Provider Comparison**: We conducted experiments across multiple LLM providers (including Anthropic, DeepSeek, Google, OpenAI, and others) to assess whether capabilities varied by model architecture and training approach. See Table 2 for the list of included models with version numbers.

4.2 Scoring framework

Model outputs were inspected along three categories:

- **Visual quality** – purely graphical concerns, including axis placement, tick formatting, label legibility, colour or marker choices, and general adherence to design best-practice.
- **Data correctness** – verification that the figure reflects the prompt: correct columns must be chosen, filters must be in line with any stated conditions, and aggregations (e.g., sums or year-on-year values) must be applied exactly once.
- **Code reliability** – assessment of the Python script itself: it should run without errors, operate on the provided `df` rather than hard-coded literals, and avoid redundant computations.

To keep grading consistent across 200 prompt–model pairs we converted each requirement into a Yes/No item, yielding a short binary checklist for every category. Binary scoring was specifically selected as it limits subjectivity, speeds up evaluation, and lets us compute average hit-rates per model with simple aggregation.

Examples include "Are the axis labels clear and readable?" (visual), "Does the filtering process correctly reflect the user's intent?" (data), and "Does the code execute without syntax errors and successfully generate a graph?" (code). An answer of *Yes* scores 1, *No* scores 0. Then, category totals are normalised to scores out of 10 to obtain visual, data, and code sub-scores, which are then combined into an overall quality score. A short version of the complete checklist used in our study is shown in Table 1. For the full version, please refer to the appendix at https://anonymous.4open.science/r/llm_visualization_results-2F67/.

5 Results

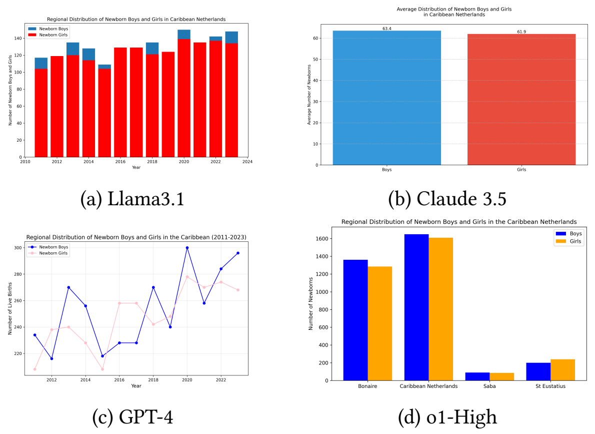
Before presenting the overall scores in Section 5.3, we make these results more tangible by first presenting a few example outcomes in Section 5.1 and then an overview of various problems that occurred in the outputs in Section 5.2.

5.1 Case Study

Figure 6 shows the results from four base (non-agentic) LLMs to the same prompt, illustrating how different LLMs interpret the same task. This prompt required models to compare the number of newborn boys and girls across different regions in the Caribbean Netherlands. We briefly discuss in which ways their outputs fail or succeed, and how visual, code, and data errors are reflected in the scoring framework.

Llama3.1 generated a bar chart with two traces overlayed over each other. The bar representing boys is not visible when smaller in value than girls. More critically, the model misinterpreted the grouping intent of the prompt by aggregating over years rather than regions, likely defaulting to temporal logic when encountering time-related fields. This resulted in deduction of scores due to incorrect filtering and column selection.

Claude 3.5 executed the code correctly and avoided syntax issues, but failed to aggregate the data by region entirely. Instead, it produced a single averaged result for the entire Caribbean Netherlands, overlooking geographic distinctions explicitly requested in the prompt.



(a) Llama3.1 (b) Claude 3.5

(c) GPT-4 (d) o1-High

Figure 6: Visualizations generated by different LLMs for the prompt: *"Plot the regional distribution of newborn boys and girls in Caribbean."*

GPT-4 introduced two major issues: it aggregated by year instead of region and used a line chart for categorical data, which was not an appropriate encoding. These choices resulted in visual misalignment and semantic mismatch.

o1-High, by contrast, successfully aggregated the data by region and plotted grouped bars for boys and girls side-by-side, ensuring both visibility and comparability, achieving the highest score in all three categories. The evaluation table detailing category-wise scoring for this prompt is included in Table 1.

5.2 Qualitative Results

Beyond the specific issues highlighted in the case study, a broader analysis of outputs, particularly from zero-shot generation attempts without iterative feedback, reveals consistent challenges across models and prompt types. These recurring problems, categorized into visual design, code implementation, and data logic, underscore the limitations of unassisted LLM-based visualization and motivated the development of more sophisticated approaches.

5.2.1 Visual Representation Issues. A frequently observed issue was the misuse of categorical x-axes, especially when plotting the `Periods` column, which is stored as strings. This led to disordered axes and uneven spacing since plotting libraries like `matplotlib` do not automatically interpret strings as temporal data. Another common flaw was failing to anchor the y-axis at zero, resulting in graphs that exaggerated minor fluctuations.

LLMs also tended to add redundant elements like both lines and markers in the same plot, which cluttered the visual and reduced clarity. Furthermore, strong gridlines and dense background structures were often included by default, diminishing the visual focus on the data. These patterns suggest that while models can produce syntactically valid plots, they lack intuition for effective visual communication.

5.2.2 Code Generation Issues. Several LLMs incorrectly attempted to convert non-standard date strings into datetime objects, causing errors during execution. A broader issue was the tendency to guess column names or refer to nonexistent fields due to the lack of

Table 1: Case study evaluation results.

Criteria	Llama	Claude	GPT-4	o1
Visual Quality Scores				
X-axis correct	1	1	1	1
Y-axis correct	1	1	1	1
Axis labels clear	1	1	1	1
Color used well	1	1	1	1
Legend accurate	1	1	1	1
Good scaling	1	0	0	1
Marks correct	0	1	1	1
Readable layout	1	1	1	1
Total	**7**	**7**	**7**	**8**
Code Quality Scores				
Correct imports	1	1	1	1
Code runs	1	1	1	1
Correct columns	0	1	1	1
Filters correctly	0	1	1	1
No hardcoding	1	1	1	1
Prompt fully handled	0	1	0	1
No redundancy	1	1	1	1
Total	**4**	**7**	**6**	**7**
Data Accuracy Scores				
Correct chart type	1	1	0	1
Column selection	0	1	1	1
Correct filtering	1	0	0	1
Correct aggregation	1	0	0	1
Subset accurate	0	1	1	1
Handles nulls	0	1	1	1
Prompt fully covered	0	1	0	1
Total	**3**	**5**	**3**	**7**

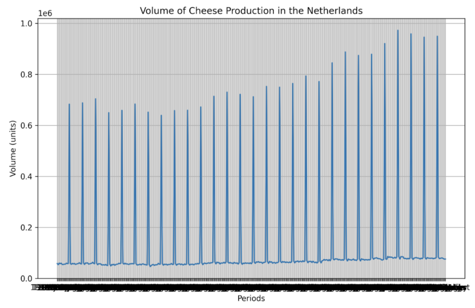


Figure 7: Visualization with poor axis sorting and excessive gridlines—typical issues seen across several models.

direct data inspection. Hardcoding values—especially prevalent in models like Gemini 2.0 Flash Thinking—was another severe flaw, undermining prompt flexibility and generalization.

Filtering logic was often mishandled, either by applying incorrect filters or omitting them altogether. These shortcomings stem largely from the model's inability to validate its assumptions against real data, leading to brittle code that may run but not meet the task requirements.

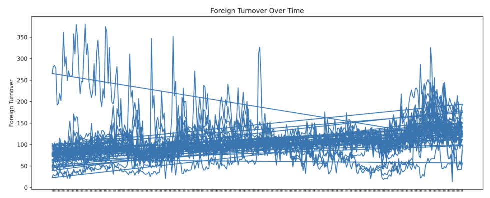


Figure 8: A case where failure to aggregate on the `Periods` column results in overlapping lines.

5.2.3 Data Representation Issues. Some models selected inappropriate chart types—for instance, using line graphs where bar charts were more suitable. Missing aggregations also led to visual noise, especially when plotting multi-period or multi-region data without combining values logically.

In other cases, null values and outliers were left unprocessed, resulting in jagged or misleading trends. Additionally, axis labels sometimes contained raw codes or identifiers from the dataset, making the plots less accessible for human interpretation.

5.2.4 Issues Beyond LLM Capabilities. Not all problems stem from the language models themselves. Some are rooted in the default behavior of tools like `matplotlib`, which often produce dense grid structures or fail to sort string-based axes. Others arise from the structure of the dataset, such as long column names with embedded indices or special characters that are difficult to reference programmatically.

These issues underscore the importance of pre-processing steps, such as renaming columns, cleaning missing values, or formatting dates before passing data to the model. More intelligent defaults in plotting libraries or clearer prompt guidance could potentially mitigate these problems.

These issues highlight the need for mechanisms that allow LLMs to refine their outputs, validate assumptions, and adhere more closely to visualization best practices.

5.3 Quantitative Results

To systematically assess LLM performance and the impact of different prompting strategies and system designs, we conducted a quantitative evaluation. We evaluated 11 model configurations, including 8 base LLMs and 3 improved variants, across 25 prompts. The prompts are categorized into 7 Easy (involving single-variable plots), 11 Medium (including multiple variables or filtering steps), and 7 Hard (requiring more complex logic and multi-dimensional comparisons) difficulty levels. The evaluation criteria involved 22 binary questions, which are grouped into three main dimensions: Visual (8 questions), Code (7 questions), and Data (7 questions). Each question was scored with a 1 (Yes) or 0 (No), and category scores were normalised to a 10-point scale, as discussed earlier.

Table 2 presents the aggregated results for all models and configurations. Across the base models, code generation was the strongest category. All models except Gemini 2.0 Flash Thinking scored above 8.0. Notably, Qwen 2.5, with 7B parameters, achieved an impressive 8.63, outperforming much larger models like Claude 3.5 (8.23) and GPT-4 (7.94). The best-performing base model in this category was o1-High with 8.97. This suggests that model size alone does not

Table 2: Normalised evaluation scores (out of 10), averaged over the 25 prompts, for each model and configuration.

Model/Configuration	Visual	Code	Data
Llama3.1 (9B)	*5.95*	8.00	*4.86*
Gemma 2 (9B)	*5.55*	8.23	*4.91*
Qwen 2.5 (7B)	5.85	8.63	5.71
Claude 3.5	6.90	8.23	6.69
Deepseek-Chat	7.00	8.57	6.74
Gemini 2.0 Flash Thinking	7.05	7.77	6.00
GPT-4o	6.20	7.94	6.00
o1-High	7.50	**8.97**	**7.37**
o1-High + Context	**7.80**	8.91	7.03
Claude 3.7 + Feedback Loop	8.90	**9.83**	**9.43**
Claude 3.7 + Feedback + Context	**9.05**	9.71	**9.43**

determine code reliability; smaller models can compete effectively when pretrained on high-quality code data.

Data handling proved to be the most challenging dimension for all of the models. The two smallest models, Llama3.1 (9B) and Gemma 2 (9B), had the weakest performance here, scoring just 4.86 and 4.91, respectively. These errors often arose from missing or incorrect filters, grouping mistakes, or improper column selection. In contrast, o1-High achieved the top base score of 7.37, followed by Deepseek-Chat (6.74) and Claude 3.5 (6.69), showing a stronger ability to manipulate structured datasets.

For most models we observed a performance drop as prompt difficulty increased, especially in the data reasoning aspect. Only Claude 3.5 and Deepseek-Chat deviated from this trend, with higher scores on more complex prompts, indicating better generalization under increasing logical complexity. More details on the breakdown of scores based on difficulty can be found in the appendix at https://anonymous.4open.science/r/llm_visualization_results-2F67/.

Visual output quality varied most widely across models. Gemini (7.05), Deepseek-Chat (7.00), and o1-High (7.50) produced mostly consistent and well-formatted charts. In contrast, Llama3.1 and Gemma 2 scored poorly in this category (5.95 and 5.55), often due to overlapping labels, incorrect axes, or inappropriate chart types, such as using line charts for categorical comparisons, as shown in the case study.

Adding structured context to the prompt, covering visualization principles, common pitfalls, and a checklist led to an improvement in o1-High's visual score, increasing from 7.50 to 7.80. However, its code and data scores slightly declined to 8.91 (from 8.97) and 7.03 (from 7.37). This suggests that while visual clarity improved, the added instructions may have introduced complexity that confused the model in areas requiring precise logic or filtering.

Enabling Claude 3.7 to revise its own outputs through a 25-step feedback loop resulted in dramatic improvements across all categories. Its visual, code, and data scores jumped to 8.90, 9.83, and 9.43, respectively—among the highest in the evaluation. The model benefitted from the ability to inspect its own plots, correct execution errors, and adapt its strategy over time.

Combining both enhancements (prompt context and self-feedback) pushed Claude's visual score even further to 9.05, while maintaining high performance in code (9.71) and data (9.43). This final configuration produced the best overall scores in the study, demonstrating

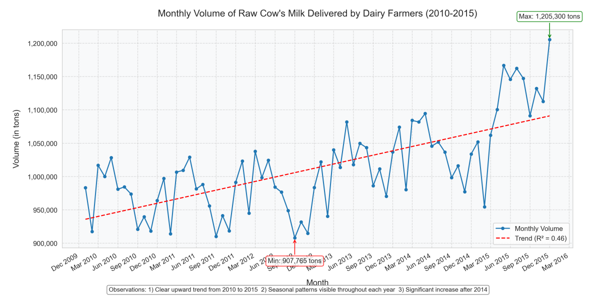


Figure 9: *"Plot the monthly volume of raw cow's milk delivered by dairy farmers between 2010–2015".* Generated by Claude 3.7 with feedback loop and contextual prompt guidance. The line chart accurately captures seasonal fluctuations. It also includes a red trendline, annotated min and max values, and an interpretive legend summarizing observed patterns.

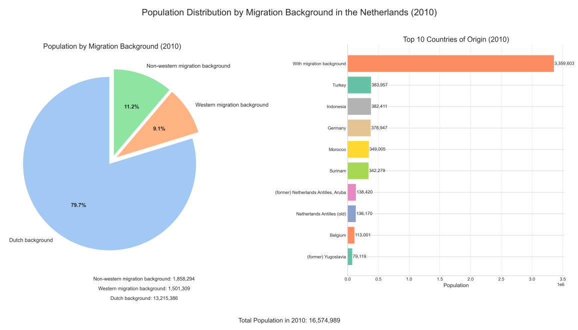


Figure 10: *"Plot population distribution by migration background in the year 2010."* This dual-plot visualization includes a pie chart and horizontal bar chart for complementary perspectives. It clearly distinguishes the main population groups and ranks the top 10 origin countries by population with precise values, labels and consistent color coding.

that iterative refinement and additional context in prompts could be used to reinforce the performance of the models. Two of such visualizations are provided in Figures 9 and 10.

In summary, these findings highlight several key observations:

- All base models perform reasonably well in code generation, but smaller models (7B–9B) struggle with data logic and visual design.
- o1-High consistently outperforms other base models, achieving the highest scores in visual, code, and data dimensions.
- Adding prompt-level context improves visual presentation, though it may slightly reduce code or data accuracy.
- Iterative feedback mechanisms enable substantial gains, especially for complex prompts requiring layered reasoning.
- When self-feedback is combined with context guidance, models like Claude 3.7 reach near-perfect scores, indicating the full potential of LLM agents in visualization tasks.

Full analysis table for each model (per question, per prompt) is included in the appendix at https://anonymous.4open.science/r/llm_visualization_results-2F67/.

6 Conclusions & Discussion

This study evaluated the capability of large language models to generate data visualizations from official statistics data. Through systematic experiments with eight LLMs across 25 tasks, we found that while base models, under one-shot conditions, achieve adequate code generation (mean score: 8.3/10), they exhibit substantial deficiencies in data manipulation (5.9/10) and visual design (6.5/10). The implementation of an agentic system with iterative self-correction yielded dramatic improvements, with Claude 3.7 achieving scores exceeding 9.0/10 across all three dimensions.

While the visualizations generated by our agentic system nearly saturate our benchmark, they suffer from an increase in complexity of the produced charts that our evaluation did not explicitly take into account. Care needs to be taken that these efforts by LLMs to iterate on their own results do not push their results outside of the grasp of the intended audience.

Beyond these technical findings, this work makes several broader contributions. We introduce a reusable evaluation framework for systematically assessing LLM performance across visualization tasks of varying complexity. By separating evaluation into three dimensions (code, data, visual), our approach provides a replicable template for future benchmarking efforts in the field. The generic nature of the agentic approach developed in this study suggests broader applicability beyond the public data of Statistics Netherlands. The modular prompt engineering, evaluation framework, and iterative self-correction mechanisms are domain-agnostic and could potentially be extended to other statistical agencies, corporate dashboards, scientific publications, and educational contexts.

The findings also offer actionable insights into how LLMs reason through structured tasks, highlighting where modifications in prompt engineering or model design could yield improvements. More broadly, this research contributes to the growing field of natural language interfaces (NLI) to data visualization—also referred to as text-to-vis—by demonstrating how advanced language models can make complex data more accessible to a broader audience.

It could be studied in further detail whether the generated visualizations are a good fit for a non-expert audience. Conducting comparative user studies between LLM-generated visualizations and those produced by Statistics Netherlands experts would provide insights into practical comprehensibility for non-expert users. Such studies could establish whether the technical sophistication we observed enhances or hinders the democratization of official statistics access.

References

[1] T. De Bie, L. De Raedt, J. Hernández-Orallo, H. H. Hoos, P. Smyth, and C. K. I. Williams. 2022. Automating data science. *Commun. ACM* 65, 3 (2022), 76–87. doi:10.1145/3495256

[2] N. Chen, Y. Zhang, J. Xu, K. Ren, and Y. Yang. 2025. VisEval: A Benchmark for Data Visualization in the Era of Large Language Models. *IEEE TVCG* 31, 1 (2025), 1301–1311. doi:10.1109/TVCG.2024.3456320

[3] Q. Chen, F. Sun, X. Xu, Z. Chen, J. Wang, and N. Cao. 2022. VizLinter: A Linter and Fixer Framework for Data Visualization. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (2022), 206–216. doi:10.1109/TVCG.2021.3114804

[4] V. Dibia. 2023. LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models. In *Proc. of ACL Demo's*. 113–126. doi:10.18653/v1/2023.acl-demo.11

[5] V. Dibia and Ç. Demiralp. 2018. Data2VIS: Automatic Generation of Data Visualizations Using Sequence-to-Sequence Recurrent Neural Networks. *IEEE CG&A* 39, 5 (2018), 33–46. doi:10.1109/MCG.2019.2924636

[6] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo. 2019. VizML: A Machine Learning Approach to Visualization Recommendation. In *Proc. of ACM CHI*. Paper No. 128. doi:10.1145/3290605.3300358

[7] J. Kouwenhoven, L. Lageweg, and B. Kruit. 2024. Constrained LLM-Based Query Generation for Question Answering on Official Statistics. *Frontiers in Artificial Intelligence and Applications* (2024), 4586–4593. doi:10.3233/faia241052

[8] L. Lageweg and B. Kruit. 2024. Generative expression constrained Knowledge-Based decoding for open data. In *Proc. of ESWC*. 307–325. doi:10.1007/978-3-031-60626-7_17

[9] S. Li, X. Chen, Y. Song, Y. Song, and C. Zhang. 2025. Prompt4Vis: Prompting Large Language Models with Example Mining and Schema Filtering for Tabular Data Visualization. *The VLDB Journal* 34 (2025), Article no. 38. doi:10.1007/s00778-025-00912-0

[10] P. Maddigan and T. Susnjak. 2023. Chat2VIS: Generating Data Visualizations via Natural Language Using ChatGPT, Codex and GPT-3 Large Language Models. *IEEE Access* 11 (2023), 45181–45193. doi:10.1109/ACCESS.2023.3274199

[11] A. Narechania, A. Srinivasan, and J. Stasko. 2021. NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries. *IEEE TVCG* (2021). doi:10.1109/TVCG.2020.3030378

[12] OECD. 2024. *Facts not Fakes: Tackling Disinformation, Strengthening Information Integrity*. Technical Report. OECD. doi:10.1787/d909ff7a-en

[13] X. Qin, Y. Luo, N. Tang, and G. Li. 2018. DeepEye: Visualizing Your Data by Keyword Search. In *Proc. of EDBT*. 441–444. doi:10.5441/002/edbt.2018.42

[14] G. Karthick S. Ram and V. Muthumanikandan. 2024. Visistant: A Conversational Chatbot for Natural Language to Visualizations With Gemini Large Language Models. *IEEE Access* 12 (2024), 138547–138563. doi:10.1109/ACCESS.2024.3407060

[15] Wonduk Seo, Seungyong Lee, Daye Kang, Hyunjin An, Zonghao Yuan, and Seunghyun Lee. 2025. Automated Visualization Code Synthesis via Multi-Path Reasoning and Feedback-Driven Optimization. doi:10.48550/arXiv.2502.11140

[16] Y. Tian, W. Cui, D. Deng, X. Yi, Y. Yang, H. Zhang, and Y. Wu. 2025. ChartGPT: Leveraging LLMs to Generate Charts from Abstract Natural Language. *IEEE TVCG* 31, 3 (2025), 1731–1745. https://doi.org/10.1109/TVCG.2024.336862

[17] P. Vázquez. 2024. Are LLMs ready for Visualization?. In *Proc. of PacificVis*. 343–352. doi:10.1109/PacificVis60374.2024.00049

[18] H. Voigt, N. Carvalhais, M. Meuschke, M. Reichstein, S. Zarrie, and K. Lawonn. 2023. VIST5: An Adaptive, Retrieval-Augmented Language Model for Visualization-Oriented Dialog. In *Proc. of EMNLP Demo's*. 70–81. doi:10.18653/v1/2023.emnlp-demo.5

[19] L. Wang, S. Zhang, Y. Wang, E. Lim, and Y. Wang. 2023. LLM4Vis: Explainable Visualization Recommendation using ChatGPT. In *Proc. of EMNLP (Industry Track)*. 675–692. doi:10.48550/arxiv.2310.07652

[20] Q. Wang, Z. Chen, Y. Wang, and H. Qu. 2022. A Survey on ML4VIS: Applying Machine Learning Advances to Data Visualization. *IEEE TVCG* (2022), 5134–5153. doi:10.1109/TVCG.2021.3106142

[21] Z. Yang, Z. Zhou, S. Wang, X. Cong, X. Han, Y. Yan, Z. Liu, Z. Tan, P. Liu, D. Yu, Z. Liu, X. Shi, and M. Sun. 2024. MatPlotAgent: Method and Evaluation for LLM-Based Agentic Scientific Data Visualization. In *Proc. of ACL*. doi:10.18653/v1/2024.findings-acl.701