Evaluating the Performance of Large Language Models in Automated Data Visualization

Anonymous ACL submission

Abstract

We present a large-scale empirical benchmark evaluating nine large language models (LLMs) on the task of automated data visualization from tabular datasets. Our benchmark comprises **4,319 fully stateless, independent runs**[1] spanning 9 models from 5 providers, 8 datasets (5 well-known, 3 unconventional), 2 prompt complexity levels, and 30 independent trials per model-dataset-prompt configuration. We introduce 11 quantitative metrics covering execution success, data fidelity, hallucination rate, visual quality, code complexity, cross-run consistency, and cost-efficiency. Our key findings are: (1) GPT-4o-mini achieves the highest overall execution success rate (**97.3%**, with Opus at 96.9%) through conservative code generation; (2) Claude Sonnet 4-5 exhibits a dramatic prompt-sensitivity reversal, dropping from 96.2% with rich prompts to 33.3% with minimal prompts, a **62.9 percentage point** gap that is the largest effect in our study; (3) DeepSeek Chat achieves 90.4% execution success with the highest data fidelity (84.3%) at near-zero cost; (4) open-source models trail non-anomalous closed-source counterparts by over 30 percentage points on average (41.5–66.9% vs. 90.4–97.3% excluding the Sonnet outlier); and (5) no model demonstrates a systematic advantage on well-known versus unconventional datasets, providing no evidence that training-data memorization is a primary driver of visualization performance. All experimental code and configurations will be released upon acceptance.

1 Introduction

The emergence of code-generating LLMs has opened a practically important question: can these models serve as autonomous data visualization agents (5)? Tabular data is everywhere in industry and research, and the ability to go from a raw Comma-Separated Values (CSV) file to an informative, well-labeled chart, with no human intervention, would substantially accelerate data analysis workflows.

General-purpose code generation benchmarks such as HumanEval (4) and MBPP (1) measure function-level programming proficiency, but they treat all code as equivalent. Data visualization is a qualitatively different task. A successful visualization requires the model to simultaneously: (a) comprehend a dataset's schema from minimal context, (b) select an appropriate chart type for the data, (c) generate syntactically and semantically correct Python code, (d) produce a fully labeled and readable output image, and (e) avoid hallucinating column names or data relationships that do not exist in the source data. Failure at any one of these steps produces an unusable result.

Despite this complexity, visualization-specific evaluation of LLMs remains understudied. Existing benchmarks focus either on natural language to query translation (9), chart comprehension (10), or single-model pipeline studies (5). None provide a systematic, multi-model, multi-dataset comparison with automated metrics across the full end-to-end visualization pipeline.

We address this gap with a comprehensive LLM visualization benchmark. Our evaluation spans **4,319 independent runs** across 9 LLMs from 5 providers, 8 tabular datasets spanning diverse domains and schema complexities, 2 distinct prompt conditions, and 30 independent trials per configuration. We introduce 11 quantitative metrics that assess the full spectrum of visualization quality, from binary execution success to perceptual color fidelity and code-level hallucination.

The scale and systematic design of our benchmark yield several surprising findings. The model with the highest execution success rate (GPT-4o-mini, 97.3%) achieves this through intentionally simple code generation. The highest-quality visu-

[1] One Gemini 2.5 Pro trial returned a 503 API error and is excluded, yielding 4,319 of 4,320 expected runs.

alizations come from models that write more complex, multi-panel code (Claude Opus), at a marginal execution cost but a dramatic cost increase (416× more expensive per successful chart). Most strikingly, Claude Sonnet 4-5 performs at Opus-level with rich prompts but drops to 33.3% with minimal context, a finding with direct implications for real-world deployment. We also confirm that training data memorization does not meaningfully drive visualization performance: models perform comparably, and often better, on datasets they almost certainly have not seen during training.

Contributions:

- A large-scale, fully automated benchmark of 4,319 stateless runs evaluating 9 LLMs on data visualization across 8 datasets and 2 prompt conditions.
- An 11-metric evaluation framework covering execution, data fidelity, hallucination, visual quality, code complexity, and cost-efficiency.
- Systematic prompt sensitivity analysis, revealing a 62.9 percentage points (pp) success rate reversal for Claude Sonnet 4-5, the most extreme instance of prompt sensitivity we are aware of in the visualization literature.
- Empirical evidence finding no evidence that training-data memorization drives LLM visualization performance.
- A cost-efficiency analysis demonstrating a 30+ percentage point performance gap between open-source and closed-source models under identical conditions.

2 Related Work

2.1 LLM Code Generation

Systematic evaluation of LLM coding ability began with function-level benchmarks. Chen et al. (4) introduced HumanEval, measuring pass@k on 164 handwritten Python problems evaluated by unit tests. Austin et al. (1) scaled this to 974 crowd-sourced problems with MBPP. Subsequent work extended code evaluation to repository-level tasks (6), multi-language settings (3). Concurrently, Wu et al. (15) applied LLM prompting to visualization recommendation, demonstrating few-shot performance competitive with supervised models. While these benchmarks are well-established, they do not account for the domain-specific requirements of data visualization: schema comprehension, chart type selection, and the avoidance of hallucinated data references.

2.2 Text-to-Visualization

The problem of converting natural language to visual representations has been studied under several related framings. NL4DV (12) maps analytical queries to Vega-Lite specifications using a grammar-guided approach. nvBench (9) provides a benchmark for natural-language-to-visualization over structured databases. Dibia (5) proposed LIDA, an LLM-powered framework that generates and iteratively refines visualizations. These systems are largely single-model or pipeline-based; our work provides the first systematic comparison across nine models under identical conditions.

2.3 Chart Understanding and Visual Question Answering

A parallel body of work evaluates model performance in understanding existing charts rather than generating them. ChartQA (10) and PlotQA (11) provide question-answering benchmarks over chart images. FigureQA (7) examines reasoning over synthetic figures. Our work differs fundamentally in direction: we evaluate generation, not comprehension, and our unit of evaluation is executable Python code with a corresponding output image rather than a natural language answer.

2.4 LLM Evaluation Frameworks

HELM (8) and BIG-bench (13) represent large-scale multi-task evaluation frameworks that measure general LLM capability across diverse scenarios. Neither framework includes data visualization as a distinct evaluation category. Our benchmark is complementary to these efforts, providing domain-specific metrics tailored to the visualization task.

2.5 Prompt Engineering and Sensitivity

The sensitivity of LLM outputs to prompt formulation is well-documented. Wei et al. (14) showed that chain-of-thought prompting substantially improves multi-step reasoning. Brown et al. (2) demonstrated that few-shot in-context examples unlock latent capabilities not apparent in zero-shot settings. In code generation specifically, prompt quality has been shown to have significant effects on pass rates (4). Our study directly measures prompt sensitivity in the visualization domain, discovering a 62.9pp gap for the most sensitive model, substantially larger than reported in general code generation settings.

Table 1: Models evaluated across 5 providers.

Model	Provider	Type	Cost/Run ($)
GPT-4o	OpenAI	Closed	0.0043
GPT-4o (16)-mini	OpenAI	Closed	0.00024
Claude Opus 4-5 (17)	Anthropic	Closed	0.100
Claude Sonnet 4-5	Anthropic	Closed	0.022
Gemini 2.5 Pro (18)	Google	Closed	0.0079
Gemini 2.5 Flash	Google	Closed	0.00017
DeepSeek Chat (19)	DeepSeek	Closed†	0.00043
Qwen3-32B	Alibaba/Groq	Open	0.00
Llama 3.1 8B (20)	Meta/Groq	Open	0.00

†DeepSeek model weights are publicly available under an MIT license. We classify it with closed-source models because we access it exclusively via its proprietary API, matching the operational conditions of other closed models in this study.

Table 2: Evaluation datasets. Known = likely in training data. Unconventional = niche/synthetic.

Dataset	Type	Rows	Cols	Domain
Titanic	Known	891	15	Survival
Iris	Known	150	5	Classification
Sleep Health	Known	374	13	Health
Customer Segments	Known	2,240	29	Marketing
Credit Fraud	Known	1,000	4	Finance
Dogs of Zurich	Unconventional	103,618	35	Municipal
Cameras	Unconventional	1,000	10	Electronics
Coffee Sales	Unconventional	2,000	10	Retail

3 Methodology

3.1 Evaluation Protocol

Every run in our benchmark is completely stateless. A fresh Application Programming Interface (API) client is instantiated for each call, and exactly one system message and one user message are issued per request. Generated code uses matplotlib (21) for rendering. Code execution occurs inside an isolated subprocess running in a temporary directory that is destroyed immediately after the run. This guarantees zero cross-contamination between all 4,319 runs. Each model-dataset-prompt configuration is repeated **30 independent times** to enable statistical characterization of run-to-run variance.

3.2 Models

We evaluate nine LLMs spanning five providers: two from OpenAI, two from Anthropic, two from Google, one from DeepSeek, and two open-source models served via Groq's inference API. Table 1 lists all models with their approximate per-run cost. Temperature is fixed at 0.7 for all models where configurable; maximum output tokens is set to 2,048 for all models; no model-specific prompt tuning is applied.

3.3 Datasets

We evaluate on eight tabular datasets split into two categories. *Known* datasets are widely used in the machine learning literature and are likely represented in LLM training corpora. *Unconventional* datasets are niche, domain-specific, or synthetic, selected to test generalization beyond potential memorization. Table 2 summarizes all eight datasets.

Datasets range from 4 to 35 columns and from 150 to 103,618 rows, providing meaningful variation in schema complexity. Customer Segments (29 columns) and Dogs of Zurich (35 columns) represent the most demanding cases for schema comprehension.

3.4 Prompt Conditions

We use two prompt conditions to measure the effect of contextual richness on model performance. The exact prompt text is identical across all models for a given dataset-condition pair. The **Rich** prompt provides full schema, description, three sample rows, and explicit file-saving directives. The **Poor** prompt provides only the first five rows of raw CSV with no schema, description, or file-handling guidance. Full templates are reproduced in Appendix C.

3.5 System Prompt

All nine models receive an identical system prompt requesting matplotlib or seaborn code with no surrounding explanation, unchanged across all 4,319 runs. The complete text is given in Appendix C.

3.6 Evaluation Metrics

We introduce 11 quantitative metrics organized across five categories: **Execution** (ES, GL), **Data Integrity** (FS, HR), **Visual Quality** (CDE, VE), **Code Quality** (LC, CC), **Economics** (CPR), and **Cross-Run** stability (CRCV, PSI). Full definitions of every metric appear in Appendix B (Table 7). Execution Success (ES) is the primary metric: a run is successful if and only if the generated code executes without error and produces a non-empty Portable Network Graphics (PNG) image file within the timeout. Fidelity Score (FS) measures data grounding via the fraction of real column names that appear as string literals in the generated code. Hallucination Rate (HR) is its complement, with important caveats discussed in Section A.3. Visual quality metrics (CDE, VE) are computed only on successful runs to avoid selection bias: CDE measures perceptual color variety in LAB space, VE captures information density via grayscale Shannon entropy. Generation Latency (GL) is wall-clock API response time in seconds.

3.7 Code Extraction and Sandboxed Execution

After a model responds, the pipeline strips markdown code fences (```python) to isolate the executable block. If no fence is present, the raw response is used as-is provided it contains an `import` statement. The extracted code is then executed in a sandboxed subprocess with a **60-second timeout** inside a temporary directory that contains copies of all dataset CSV files. Path references are normalized to absolute paths before execution. The temporary directory is deleted immediately after the run regardless of outcome.

3.8 Auto-Fix Protocol

When code fails due to column-not-found errors (`KeyError` or `ValueError` containing "column" or "not in index"), an automated single-pass fix is applied: actual column names are read from the source CSV, and case-insensitive regex replacement is performed on mismatched string references. The corrected code is then re-executed. The original error, the fix description, and the final outcome are all recorded in the result JSON. This protocol triggered in **60 of 4,319 runs (1.4%)**, indicating that the vast majority of failures are not recoverable by simple column-name correction.

4 Results

4.1 Overall Performance

Table 3 reports aggregate performance across all 4,319 runs ranked by execution success rate.

GPT-4o-mini achieves the highest execution success (97.3%) by generating conservative, low-complexity code (CC = 0.70). **Claude Opus** achieves near-highest data fidelity (84.2, second only to DeepSeek at 84.3) and labeling completeness (96.7%) by writing ambitious multi-panel dashboards (CC = 5.24), with only a marginal execution cost. **DeepSeek Chat** matches Opus on fidelity (84.3) at a per-run cost roughly 233 times lower.

The two open-source models represent a sharp discontinuity: Llama 3.1 8B and Qwen3-32B achieve 66.9% and 41.5% execution success respectively, compared to 90.4–97.3% for closed-source models excluding the Sonnet anomaly (Section 4.3). This 30+ percentage point gap reflects fundamental differences in code generation reliability and instruction-following behavior, not simply a matter of model scale. Figure 1 visualizes execution success across all 72 model–dataset combinations.

4.2 Failure Analysis

Table 4 breaks down the cause of every failed run. Failure modes differ markedly by model family.

GPT-4o and GPT-4o-mini fail primarily through execution timeouts, code that is syntactically correct but computationally expensive within the 60-second limit. Gemini 2.5 Flash fails predominantly through `ValueErrors` from incorrect matplotlib API parameter usage (23 of 26 failures), while Gemini 2.5 Pro shows a mix of `ValueErrors` (18) and `AttributeErrors` (17). Claude Sonnet 4-5's 169 failures are dominated by the "Other" category (144 runs), the overwhelming majority of which (130) are `FileNotFoundErrors`: the model generates syntactically valid Python in a properly formatted code block, but references a generic filename (e.g., `pd.read_csv('data.csv')`) rather than the dataset-specific path. Our pipeline's path-normalization handles explicit filenames (e.g., `titanic.csv`) but cannot resolve generic placeholders. This failure mode correlates directly with the poor-prompt condition (Section 4.3), where no explicit file path is provided. Notably, this is not a code-generation deficit: Sonnet produces clean, well-structured Python; it simply lacks the filename context that rich prompts supply. By contrast, Llama 3.1 8B, when given the same poor prompts, more frequently guesses dataset-specific filenames or embeds CSV content inline, avoiding this failure mode at the cost of lower code sophistication. Qwen3-32B's 263 "Other" failures are predominantly (232) `SyntaxErrors` resulting from code extraction failures: the model wraps its output in non-standard formatting (e.g., extended thinking tags, unusual fence variants) that our extraction regex does not fully handle. An additional 31 are `FileNotFoundErrors` analogous to Sonnet's pattern. Figure 2 illustrates these distinct failure regimes visually.

4.3 Prompt Sensitivity

Table 5 reports execution success under both prompt conditions, along with each model's Prompt Sensitivity Index (PSI).

The most striking finding in our study is Claude Sonnet 4-5's PSI of 62.9 percentage points. With a rich, well-structured prompt, Sonnet performs at the level of Claude Opus (96.2%); with a

Table 3: Overall Model Performance Scorecard (ranked by Execution Success). N = total runs per model.

Rank	Model	N	ES%	Fidelity	Halluc.%	Label%	Latency	Cost ($/run)
1	GPT-4o-mini	480	97.3	58.7	54.2	85.8	6.2s	0.00024
2	Claude Opus 4-5	480	96.9	84.2	60.7	96.7	13.5s	0.100
3	GPT-4o	480	95.4	62.0	53.8	77.7	3.7s	0.0043
4	Gemini 2.5 Flash	480	94.6	69.3	52.9	82.7	15.2s	0.00017
5	Gemini 2.5 Pro	479	90.8	73.5	65.0	87.2	30.7s	0.0079
6	DeepSeek Chat	480	90.4	84.3	59.9	91.5	26.8s	0.00043
7	Llama 3.1 8B	480	66.9	72.4	45.1	74.5	3.9s	0.00
8	Claude Sonnet 4-5	480	64.8	76.7	62.1	92.6	16.2s	0.022
9	Qwen3-32B	480	41.5	64.2	49.4	63.7	21.3s	0.00

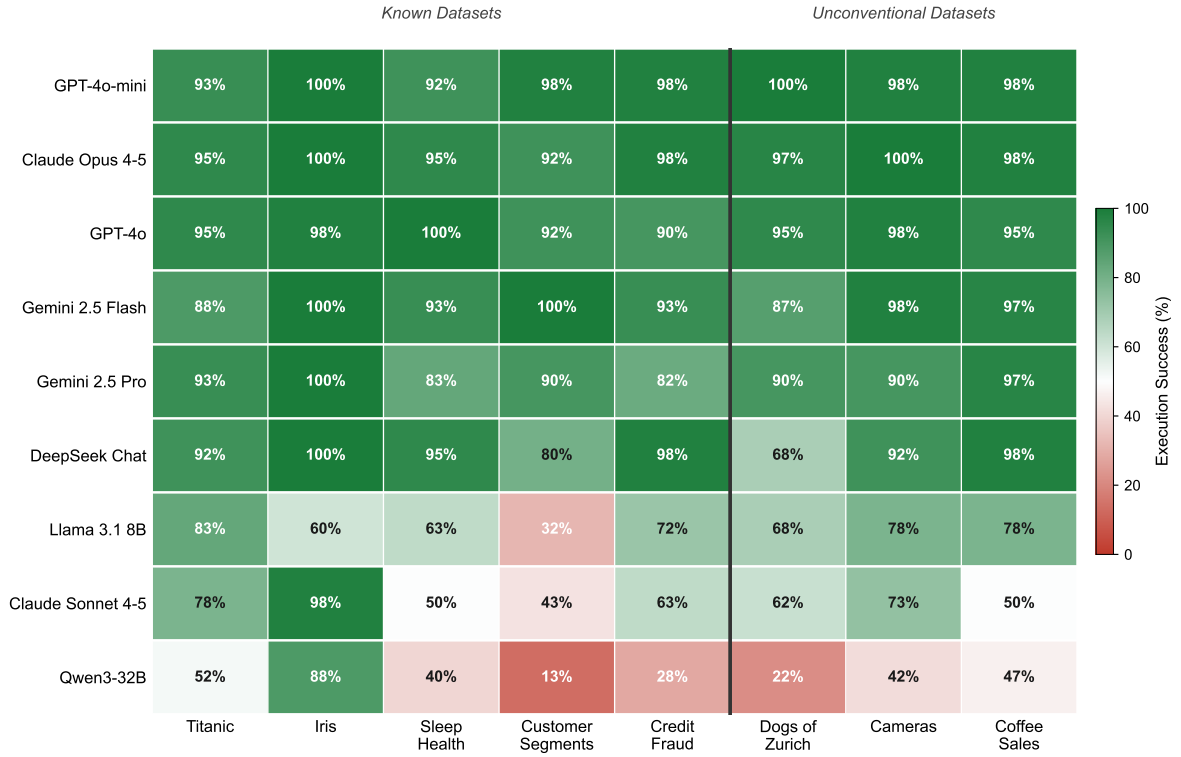

Figure 1: Execution success rate (%) for each model–dataset combination. Green indicates high success; red indicates failure. The vertical divider separates the five well-known datasets (left) from the three unconventional datasets (right). The absence of a systematic color difference across the divider refutes training-data memorization as a performance driver.

Table 4: Failure Cause Breakdown by Model. "Other" captures file path inference errors, code extraction failures, and residual non-parseable output.

Model	Fails	Timeout	KeyError	TypeError	ValueError	AttrError	Other
GPT-4o-mini	13	7	0	0	1	3	2
Claude Opus 4-5	15	0	1	3	8	0	3
GPT-4o	22	10	0	5	4	3	0
Gemini 2.5 Flash	26	0	0	1	23	1	1
Gemini 2.5 Pro	44	0	1	5	18	17	3
DeepSeek Chat	46	0	1	1	19	0	25
Llama 3.1 8B	159	58	4	12	67	13	5
Claude Sonnet 4-5	169	0	0	1	4	20	144
Qwen3-32B	281	3	0	0	0	15	263

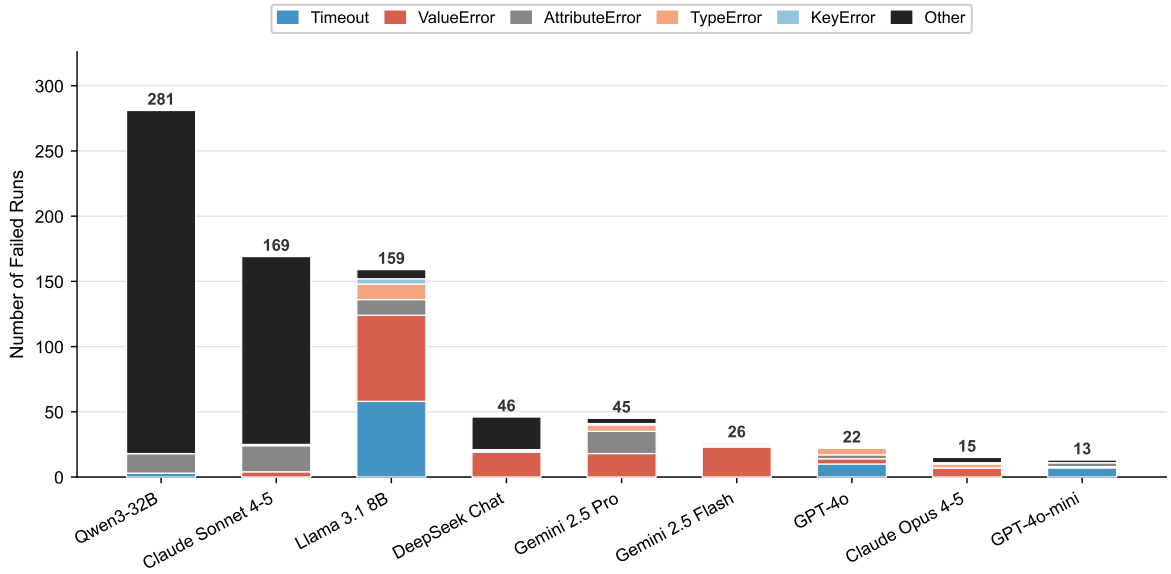

Figure 2: Failure mode breakdown per model, sorted by total failures. Qwen3-32B and Claude Sonnet 4-5 are dominated by the "Other" category (file path errors and code extraction failures), while Llama 3.1 8B exhibits a mix of technical runtime errors, and GPT-4o-mini fails almost exclusively through execution timeouts.

Table 5: Prompt Sensitivity. PSI = |Rich ES% − Poor ES%|. Negative gaps indicate better performance on poor prompts.

Model	Rich	Poor	Gap	PSI
Claude Sonnet 4-5	96.2	33.3	+62.9pp	62.9
Qwen3-32B	58.3	24.6	+33.8pp	33.8
Llama 3.1 8B	78.8	55.0	+23.8pp	23.8
GPT-4o-mini	100.0	94.6	+5.4pp	5.4
GPT-4o	97.5	93.3	+4.2pp	4.2
DeepSeek Chat	88.8	92.1	−3.3pp	3.3
Claude Opus 4-5	96.2	97.5	−1.2pp	1.2
Gemini 2.5 Pro	90.4	91.2	−0.8pp	0.8
Gemini 2.5 Flash	94.6	94.6	0.0pp	0.0

minimal poor prompt it collapses to 33.3%, a level worse than Llama 3.1 8B (55.0% on poor prompts). Failure analysis reveals a specific and consistent mechanism: 130 of Sonnet's 144 "Other" failures are `FileNotFoundErrors` caused by the model referencing a generic placeholder filename (`data.csv`) rather than the dataset-specific path. Rich prompts explicitly provide the file path (`data/titanic.csv`), which Sonnet uses correctly; poor prompts omit this information, and Sonnet defaults to a generic name that our sandbox cannot resolve. Sonnet achieves 0% execution success on poor prompts for three datasets (Coffee Sales, Customer Segments, Sleep Health), driving its overall rate below Llama. By contrast, Llama more frequently guesses dataset-specific filenames or embeds CSV data inline, avoiding this file-resolution failure at the cost of lower code sophistication.

By contrast, Claude Opus (PSI = 1.2), Gemini 2.5 Pro (PSI = 0.8), and Gemini 2.5 Flash (PSI = 0.0) are essentially invariant to prompt quality. Three models (DeepSeek Chat, Claude Opus, and Gemini 2.5 Pro) exhibit an inverted relationship: they perform marginally better with poor prompts. We interpret this as larger models inferring task intent more flexibly when given less prescriptive instructions, and being less constrained by explicit file-saving directives that smaller models occasionally mishandle.

Prompt sensitivity varies by dataset complexity. Rich prompts consistently provide the largest gains for high-column-count, complex schemas: Sleep Health (+23.3pp), Coffee Sales (+17.1pp), Customer Segments (+17.0pp), and Titanic (+16.3pp). Iris benefits least (+3.3pp), consistent with its simple 5-column structure being easily inferred from raw CSV rows alone. Figure 3 makes this divergence immediately visible.

4.4 Known vs. Unconventional Datasets

To test whether visualization performance is driven by training-data familiarity, we partition each model's runs by whether the dataset is widely used in ML literature ("known") or niche and unlikely to be memorized ("unconventional"). Full per-model results are in Appendix E (Table 9).

No systematic advantage exists for well-known datasets: five of nine models perform better on unconventional ones, four perform better on known

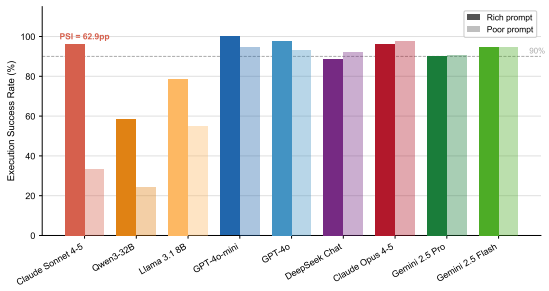

Figure 3: Execution success rate under rich prompts (solid) vs. poor prompts (faded), sorted by Prompt Sensitivity Index (PSI). Claude Sonnet 4-5 exhibits a 62.9pp reversal, while Gemini 2.5 Flash (rightmost) is completely invariant to prompt quality (PSI = 0.0pp).

ones, and the gaps are small for high-performing models (under 3pp for GPT-4o-mini, Opus, and GPT-4o). Llama 3.1 8B shows the most pronounced reversal (−13.0pp in favor of unconventional datasets), which we attribute to its near-complete failure on Customer Segments (32% ES on a 29-column schema) dragging down its known-dataset average, a schema-complexity effect rather than a memorization effect. These results provide no evidence that visualization performance is driven by training-data familiarity with specific datasets.

4.5 Visual Quality

Claude Opus produces the most professionally labeled visualizations: highest Color ΔE (1.330), labeling completeness (96.7%), and CC = 5.24 (multi-panel dashboards). A consistent pattern is that failed runs have higher CC than successful ones: more complex code introduces more failure points. The exception is Sonnet, whose failures are structural (file paths) rather than code crashes. Full breakdown in Appendix D.

4.6 Hallucination Analysis

All nine models exhibit high hallucination rates (45–65% across conditions), and critically, higher hallucination does *not* correlate with worse execution success. Claude Opus hallucinates at 60.7% yet achieves 96.9% execution success and the highest fidelity in the benchmark. The full condition-stratified breakdown by model is in Appendix F (Table 10). This apparent paradox is explained in Section A.3.

4.7 Cost-Efficiency

Table 6 presents cost per run, execution success rate, and cost per successful visualization relative to the baseline of GPT-4o-mini.

Table 6: Cost-Efficiency Analysis. Index normalized to GPT-4o-mini ($0.000249 per successful visualization).

Model	Cost/Run	ES%	Cost/Succ.	Index
Llama 3.1 8B	$0.000	66.9	$0.000	0×
Qwen3-32B	$0.000	41.5	$0.000	0×
Gemini 2.5 Flash	$0.000173	94.6	$0.000183	0.7×
GPT-4o-mini	$0.000242	97.3	$0.000249	1.0×
DeepSeek Chat	$0.000431	90.4	$0.000477	1.9×
GPT-4o	$0.004330	95.4	$0.004538	18×
Gemini 2.5 Pro	$0.007944	90.8	$0.008747	35×
Claude Sonnet 4-5	$0.021684	64.8	$0.033468	134×
Claude Opus 4-5	$0.100396	96.9	$0.103634	416×

Claude Opus costs 416× more per successful visualization than GPT-4o-mini for only +0.4pp execution success. Gemini 2.5 Flash presents the strongest Pareto outcome (0.7× baseline cost, 94.6% ES). DeepSeek Chat offers premium fidelity (84.3%) at 1.9× baseline.

4.8 Latency

GPT-4o is fastest (mean 3.7s), Llama has lowest median (1.0s) but highest variance (bimodal: fast failures vs. slow timeouts). Claude Opus is most consistent (std=2.6s). Full percentiles in Appendix G.

5 Discussion

5.1 The Capability-Ambition Trade-off

A central pattern in our results is the tension between code ambition and execution reliability. Higher-capability models (Claude Opus, DeepSeek Chat) naturally write multi-panel dashboards with complex layouts and rich annotations, producing better fidelity and visual quality. But every additional control-flow structure is a potential failure point. GPT-4o-mini's near-zero code complexity (CC = 0.70) is not a limitation; it is an architectural choice that produces the most reliably executable visualizations in the entire benchmark. Whether simple-but-reliable or complex-but-richer is preferable depends entirely on the deployment context and tolerance for failure.

5.2 The Claude Sonnet Anomaly

Claude Sonnet 4-5's PSI of 62.9pp is the most extreme prompt-sensitivity result we are aware of in the LLM evaluation literature. The failure mode is specific and consistent: with poor prompts, Sonnet generates syntactically valid Python code in a properly formatted code block, but references a generic placeholder filename (`data.csv`) rather than the dataset-specific path. This causes

a `FileNotFoundError` at runtime, accounting for 130 of its 144 "Other" failures. This is not a code-generation deficit; with rich prompts that explicitly specify the file path, Sonnet performs identically to Claude Opus (both at 96.2%). It is a structural dependence on explicit file-path context rather than on prompt scaffolding for code-generation activation *per se*. The model's code quality remains high regardless of prompt condition; what changes is its ability to infer missing operational parameters. Practitioners deploying Sonnet without carefully engineered prompts that include explicit data paths face a high probability of file-resolution failures and significant wasted API cost.

5.3 No Evidence of Training Data Memorization

The hypothesis that LLMs perform well on standard datasets primarily because they have memorized them during training is not supported by our evidence. Five of nine models perform better on unconventional datasets, and the performance gaps for strong models are too small (under 3pp) to constitute a meaningful memorization effect. The pattern is better explained by schema complexity: simpler schemas (Iris: 5 columns) are easier to handle regardless of familiarity, while complex schemas (Dogs of Zurich: 35 columns) are harder regardless of novelty. Visualization capability appears to generalize effectively from training data to domains and schemas not seen during training.

5.4 The Open-Source Performance Gap

The 30.4pp gap between the best open-source (Llama, 66.9%) and best closed-source model (GPT-4o-mini, 97.3%) reflects instruction-following alignment, not raw capability. Excluding the Sonnet anomaly, closed-source models achieve 90.4–97.3%. Open-source failures are predominantly structural: code extraction failures (Qwen) and mixed runtime errors (Llama), suggesting that alignment for reliably obeying "return ONLY the code block" is itself a significant capability threshold.

5.5 DeepSeek Chat: Optimal Low-Cost Alternative

DeepSeek Chat achieves 90.4% ES, highest fidelity (84.3%), and labeling at 91.5%, all at \$0.00043/run (233× cheaper than Opus). It exhibits an inverted fidelity pattern: poor-prompt fidelity is 99.4% vs. 72.4% on rich (computed over successful runs per condition), suggesting minimal context produces "plot what you see" code. On Dogs of Zurich (103K rows), rich prompts cause complex code that times out (12/30 vs. 29/30 poor).

6 Conclusion

We present a comprehensive benchmark of 4,319 fully stateless LLM runs evaluating 9 models on automated data visualization across 8 datasets, 2 prompt conditions, and 11 quantitative metrics, a systematic study of LLM visualization capability. Our findings are:

1. **GPT-4o-mini** achieves the highest execution success rate (97.3%) through conservative code generation (CC = 0.70), making it the most reliable choice for production visualization pipelines.
2. **Claude Opus 4-5** generates the highest-quality visualizations (fidelity 84.2, labeling 96.7%) at 416× the per-visualization cost of GPT-4o-mini, justified only when visual quality is the primary objective and cost is not a constraint.
3. **Claude Sonnet 4-5** exhibits a large 62.9pp prompt-sensitivity gap, rendering it unreliable without carefully engineered rich prompts despite its strong rich-prompt performance.
4. **DeepSeek Chat** is the optimal low-cost alternative: 90.4% execution success, the highest fidelity of any model (84.3%), and near-zero cost.
5. **Open-source models** (Llama, Qwen) lag 30+ pp behind non-anomalous closed-source counterparts, with failures rooted in code extraction and instruction-following issues rather than code quality.
6. **Training data memorization** shows no measurable effect on visualization performance: unconventional datasets perform comparably to well-known ones.
7. **Universal hallucination** (45–65% across all models) underscores the need for post-generation validation in any production deployment.

For practitioners: GPT-4o-mini and Gemini 2.5 Flash offer the best reliability-to-cost trade-offs. DeepSeek Chat is the best alternative when visualization depth is prioritized. Claude Opus maximizes visual quality when cost is unconstrained. Claude Sonnet requires explicit rich prompts to perform reliably and should not be deployed with minimal prompt context.

7 Limitations

Single-turn generation. No iterative refinement; multi-turn performance may differ. **Fixed system prompt.** An identical prompt may favor models whose tuning aligns with that phrasing. **60-second timeout.** Disadvantages models writing correct but slow code; 10/22 GPT-4o failures and DeepSeek's Dogs of Zurich inversion are timeout-related. **Path normalization.** Our pipeline patches dataset-specific filenames but cannot resolve generic placeholders (`data.csv`). This directly causes Sonnet's poor-prompt failures; its 62.9pp PSI is thus an upper bound on true prompt sensitivity for code quality. **Output token limit.** All models capped at 2,048 tokens; complex multi-panel code may be truncated. **Hallucination metric.** Our HR conflates benign string literals with genuine column-name hallucinations. **No human evaluation.** Aesthetics and chart-type appropriateness are not captured. **Single inference provider for open-source models.** Llama and Qwen are served via Groq; results may reflect deployment-specific quantization or sampling behavior rather than pure model capability.

8 Ethical Considerations

This work evaluates existing LLMs on data visualization and does not involve training new models or collecting human data. All datasets are publicly available with no PII. Automated visualization tools could generate misleading charts; our hallucination metric highlights this risk. API calls were made under each provider's terms of service during April–May 2025. All pricing figures reflect publicly listed rates as of May 2025. This paper was drafted with AI writing assistance (grammar and formatting suggestions); all scientific claims and analysis are the sole work of the authors.

References

[1] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020.

[3] F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman, A. Guha, M. Greenberg, and A. Jangda. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023.

[4] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] V. Dibia. LIDA: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*, pages 113–126, 2023.

[6] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

[7] S. E. Kahou, V. Michalski, A. Atkinson, A. Kádár, A. Trischler, and Y. Bengio. FigureQA: An annotated figure dataset for visual reasoning. In *International Conference on Learning Representations (ICLR) Workshop*, 2018.

[8] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, et al. Holistic evaluation of language models. *Transactions on Machine Learning Research (TMLR)*, 2023.

[9] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin. Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, pages 1235–1247, 2021.

[10] A. Masry, X. L. Do, J. Q. Tan, S. Joty, and E. Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics (ACL)*, pages 2263–2279, 2022.

[11] N. Methani, P. Ganguly, M. M. Khapra, and P. Kumar. PlotQA: Reasoning over scientific plots. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1516–1525, 2020.

[12] A. Narechania, A. Srinivasan, and J. Stasko. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 27(2):369–379, 2021.

[13] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, et al. Beyond

the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research (TMLR)*, 2023.

[14] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837, 2022.

[15] Lei Wang, Songheng Zhang, Yun Wang, Ee-Peng Lim, and Yong Wang. LLM4Vis: Explainable visualization recommendation using ChatGPT. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 675–692, Singapore, 2023.

[16] OpenAI. GPT-4o system card. Technical report, OpenAI, 2024.

[17] Anthropic. The Claude model family. Technical report, Anthropic, 2025.

[18] Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[19] DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[20] Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[21] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

A Additional Discussion

A.1 GPT-4o vs. GPT-4o-mini

Counterintuitively, GPT-4o-mini (97.3%) outperforms GPT-4o (95.4%) on raw execution success despite being a smaller model. GPT-4o-mini achieves a perfect 240/240 on rich prompts, while GPT-4o fails on 6 rich-prompt runs. GPT-4o's 22 failures include 10 execution timeouts (45%), indicating that the larger model generates computationally heavier operations (e.g., iterating over large DataFrames without vectorization) that exceed the 60-second limit. GPT-4o-mini, by contrast, produces slightly higher cyclomatic complexity code (CC = 0.70 vs. 0.49) but with more predictable execution patterns. This suggests that model scale alone does not determine reliability; smaller models trained for efficiency may be preferable in latency-constrained deployments.

A.2 Gemini 2.5 Flash: Complete Prompt Invariance

Gemini 2.5 Flash achieves a PSI of exactly 0.0pp (227/240 on both rich and poor prompts). Unlike Sonnet's catastrophic dependence on prompt quality, Flash maintains near-identical performance regardless of whether it receives full schema context or minimal CSV rows. Its failures (26 total) are predominantly `ValueErrors` from incorrect matplotlib API usage (23/26), distributed evenly across prompt conditions. This invariance suggests robust internal schema inference: Flash can reliably determine column types, select appropriate chart types, and generate correct file-handling code from raw CSV alone.

A.3 Hallucination Requires Careful Reinterpretation

The near-universal 45–65% hallucination rates should not be read as a red flag without qualification. Our hallucination metric (HR) counts quoted string literals in the code that do not match column names, a conservative proxy that necessarily includes variable names, style strings, color codes, chart titles, and axis labels that happen not to be column references. High-performing models produce more verbose, annotated code with more such literals, which inflates their apparent hallucination rate without affecting execution correctness. Claude Opus, which scores highest on quality (fidelity 84.2, labeling 96.7%), also has a high HR (60.7%) for exactly this reason. Future work should develop finer-grained metrics that distinguish between genuine column-name hallucinations and benign string usage.

B Evaluation Metric Definitions

Table 7 reproduces the full definitions of every metric used in the benchmark.

C Full Prompt Templates

System Prompt.

```
You are an expert data
visualization specialist.
Generate Python code using
matplotlib/seaborn to create a
clear, insightful visualization.
Return ONLY the Python code
block, no explanation, no
markdown text outside the code
block.
```

Table 7: Evaluation metrics and definitions.

Category	Metric	Definition
Execution	ES	Binary: code runs and produces a valid PNG
	GL	Wall-clock API response time (seconds)
Data Integrity	FS	% of real column names referenced in generated code
	HR	Fraction of quoted string literals not matching any real column name
Visual Quality	CDE	Mean perceptual color difference between adjacent pixels in LAB color space
	VE	Shannon entropy of the pixel intensity distribution
Code Quality	LC	Fraction of labeling elements present (xlabel, ylabel, title, legend)
	CC	Cyclomatic proxy: count of if/for/while/try/except constructs
Economics	CPR	Estimated cost based on input/output token counts
Cross-Run	CRCV	Coefficient of variation of fidelity score across 30 runs
	PSI	$\lvert$Rich ES% $-$ Poor ES%$\rvert$

Rich Prompt.

```
Dataset: {name} | Description:
{description} | Shape:
{rows}×{cols} | Columns:
{col: dtype ...} | Sample
(3 rows): {csv_head} |
Task: Create an insightful,
well-labeled visualization.
Save as output.png using
plt.savefig('output.png',
bbox_inches='tight', dpi=150).
Do not use plt.show().
```

Poor Prompt.

```
Visualize this dataset and
save the result as output.png.
{first 5 rows as raw CSV only}
```

D Visual Quality and Code Complexity

E Known vs. Unconventional Dataset Performance

F Hallucination Rates by Condition

G Response Latency Percentiles

Table 8: Visual Quality and Code Complexity on Successful Runs. CC (fail) illustrates the complexity-failure relationship. Visual quality metrics (CDE, VE) require a successfully generated image.

Model	N	Color ΔE	Entropy	Label%	CC (succ.)	CC (fail)
Claude Opus 4-5	465	1.330	1.603	96.7	5.24	6.07
Claude Sonnet 4-5	311	1.288	1.598	95.3	4.36	2.64
DeepSeek Chat	434	0.811	1.822	91.7	3.90	8.07
Gemini 2.5 Pro	435	0.776	1.451	86.7	6.67	8.61
GPT-4o-mini	467	1.033	1.238	85.6	0.70	0.69
Gemini 2.5 Flash	454	1.050	1.217	83.2	6.11	8.27
GPT-4o	458	1.068	1.286	77.8	0.49	0.73
Llama 3.1 8B	321	1.190	1.118	77.6	0.83	0.81
Qwen3-32B	199	1.212	1.334	62.2	0.07	28.22

Table 9: Known vs. Unconventional Dataset Performance.

Model	Known	Unc.	Finding
GPT-4o-mini	96.3	98.9	Unc. better (-2.6pp)
Claude Opus 4-5	96.0	98.3	Unc. better (-2.3pp)
GPT-4o	95.0	96.1	Unc. better (-1.1pp)
Gemini 2.5 Flash	95.0	93.9	Known better (+1.1pp)
Gemini 2.5 Pro	89.7	92.7	Unc. better (-3.1pp)
DeepSeek Chat	93.0	86.1	Known better (+6.9pp)
Claude Sonnet 4-5	66.7	61.7	Known better (+5.0pp)
Llama 3.1 8B	62.0	75.0	Unc. better (-13.0pp)
Qwen3-32B	44.3	36.7	Known better (+7.7pp)

Table 10: Hallucination Rate by Condition (%). All models hallucinate heavily (45–65%). See Section A.3 for interpretation.

Model	Overall	Rich	Poor	Known	Unc.
Llama 3.1 8B	45.1	48.2	42.0	44.4	46.2
Qwen3-32B	49.4	54.8	44.1	49.3	49.6
Gemini 2.5 Flash	52.9	54.8	51.0	55.1	49.2
GPT-4o	53.8	57.7	49.9	54.1	53.3
GPT-4o-mini	54.2	57.8	50.6	54.5	53.7
DeepSeek Chat	59.9	65.9	54.0	59.7	60.3
Claude Opus 4-5	60.7	62.1	59.3	60.8	60.5
Claude Sonnet 4-5	62.1	62.7	61.6	61.2	63.7
Gemini 2.5 Pro	65.0	65.1	64.9	67.0	61.7

Table 11: Response Latency Percentiles (seconds).

Model	p50	p90	p99	Mean	Std
GPT-4o	3.3s	5.9s	10.2s	3.7s	1.8s
Llama 3.1 8B	1.0s	11.9s	20.7s	3.9s	5.0s
GPT-4o-mini	5.1s	9.8s	16.6s	6.2s	3.2s
Claude Opus 4-5	12.9s	17.3s	20.8s	13.5s	2.6s
Gemini 2.5 Flash	12.8s	24.6s	44.9s	15.2s	18.9s
Claude Sonnet 4-5	16.4s	21.9s	25.4s	16.2s	4.3s
Qwen3-32B	22.1s	29.3s	37.4s	21.3s	6.9s
DeepSeek Chat	26.8s	38.2s	42.4s	26.8s	8.1s
Gemini 2.5 Pro	29.1s	41.8s	61.4s	30.7s	10.5s