LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models

Victor Dibia
Microsoft Research
victordibia@microsoft.com

Abstract

Systems that support users in the automatic creation of visualizations must address several subtasks - understand the semantics of data, enumerate relevant visualization goals and generate visualization specifications. In this work, we pose visualization generation as a multi-stage generation problem and argue that well-orchestrated pipelines based on large language models (LLMs) and image generation models (IGMs) are suitable to addressing these tasks. We present LIDA, a novel tool for generating grammar-agnostic visualizations and infographics. LIDA comprises of 4 modules - A SUMMARIZER that converts data into a rich but compact natural language summary, a GOAL EXPLORER that enumerates visualization goals given the data, a VISGENERATOR that generates, refines, executes and filters visualization code and an INFOGRAPHER module that yields data-faithful stylized graphics using IGMs. LIDA provides a python api, and a *hybrid* USER INTERFACE (direct manipulation and *multilingual* natural language) for interactive chart, infographics and data story generation.

1 Introduction

Visualizations make data accessible by reducing the cognitive burden associated with extracting insights from large tabular datasets. However, visualization authoring is a complex creative task, involving multiple steps. First the user must build familiarity with the dataset (content and semantics) and enumerate a set of relevant goals or hypotheses that can be addressed using the data. Next, users must select the right visualization representation (marks, transformations and layout) for each goal. Finally, the user must implement the visualization either as code or using available direct manipulation interfaces. Each of these steps require expertise, and can be tedious as well as error prone for *users with limited visualization experience* (novices). Existing research has sought to address these challenges by *automating* the visualization (AUTOVIZ) creation process, given a dataset (Podo et al., 2023). *Automation* may occur in two modes: i.) fully automated - the system automatically generates visualizations relevant to the data ii.) semi-automated - the user specifies their goals and the system generates visualizations that address these goals. The former mode is valuable for users unfamiliar with the data and the latter is valuable for users with some familiarity with the data and the visualization task.

Consequently, a successful AUTOVIZ tool must excel at each of several *subtasks* - understand the semantics of the data, enumerate relevant visualization goals and generate visualization specifications that meet syntax, design, task and perceptual requirements of these goals (Podo et al., 2023). Furthermore, given the target demographic (novice users), such a tool must support the user by offering NL (NL) interaction modalities (Mitra et al., 2022), affordances to control system behavior and sense making tools to understand and debug/verify system behavior. While related work has addressed aspects of the AUTOVIZ task, there are several known limitations (Podo et al., 2023) such as they: (i) rely on heuristics that are limited in coverage, challenging to craft and tedious to maintain (Wongsuphasawat et al., 2017). (ii) require significant user interaction to generate visualizations (Wongsuphasawat et al., 2017; Moritz et al., 2018). (iii) implement automated approaches that offer limited control over system input and output (Dibia and Demiralp, 2019) (iv) require grammar (or chart type) specific training data and model architectures (Dibia and Demiralp, 2019; Luo et al., 2018) for each sub task, (v) do not consider alternative chart representation formats such as infographics which have been shown to be more visually pleasing, engaging and memorable (Bateman et al., 2010; Haroz et al., 2015; Harrison et al., 2015).

Concurrently, advances in large foundation mod-

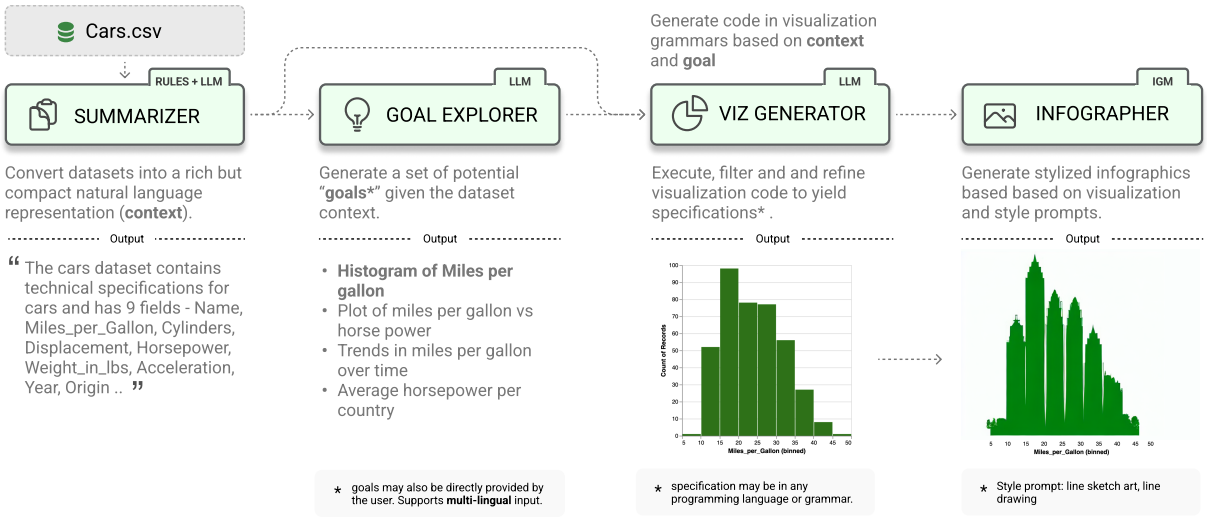

Figure 1: LIDA generates visualizations and infographics across 4 stages implemented in modules - data summarization, goal exploration, visualization generation and infographics generations. Example output from each module is shown.

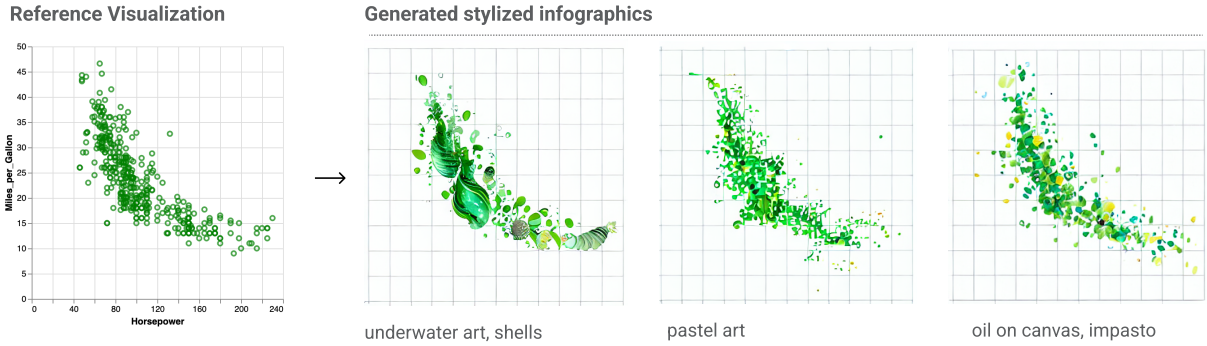

Figure 2: Example *data-faithful* infographics and associated style prompts generated with LIDA (based on generated visualization specifications).

els (Bommasani et al., 2021) have shown state of the art performance on a variety of creative tasks such as multilingual text generation, code generation, image captioning, image generation, and image editing. In this work, we argue that the vast capabilities of these models can be *assembled* to address the AUTOVIZ task, whilst *addressing the limitations of existing approaches*. We propose a multi-stage pipeline that uses LLMs[1] to generate grammar-agnostic visualization specifications and infographics and implement these ideas in a tool called LIDA. This work makes the following contributions:

- We present a novel multi-stage, modular[2] approach (Fig 1) for the automatic generation of data visualization and infographics using LLMs. Specifically, we (i) Efficiently represent datasets as NL summaries, suitable as grounding context for an LLM to address visualization tasks. (ii) Generate a set of visualization goals using LLMs. Importantly, we leverage prompt engineering to steer the model towards generating *correct* visualization that follow *best practices* (see 4). (iii) Apply LLMs to generate grammar-agnostic visualization specification based on generated (or human provided) goals. (iv) Provide a *hybrid interface* that supports traditional direct manipulation controls (e.g., manually select which fields to explore) and a rich **multilingual** NL interface to support user's with varied skill/experience. (v) Apply text-conditioned image generation models (IGM) models in generating stylized infographics that are both informative (gen-

[1]This work primarily utilizes the OpenAI *gpt-3.5-turbo-x* line of models for text and code generation.

[2]In this article, behaviors of LIDA modules are illustrated using the cars dataset (see Fig 1, 3, 4).

erally faithful to data), aesthetically pleasing and engaging. This approach also opens up new possibilities for creative visual artifacts customized to the user's preferences (color scheme and brand).

- We implement our approach in an Open Source library - LIDA[3]. LIDA provides a python api, a web api and a **rich web interface** useful for research and practical applications.

Compared to existing AUTOVIZ approaches, LIDA proposes an implementation that is simplified (eliminates the need for subtask-specific models), general (can be adapted generate visualizations in any programming language or grammar), flexible (individual modules can be optimized) and scalable (the system performance will *improve* with advances in the underlying LLM). Taken together, these contributions provide building blocks for progress towards complex workflow such as visualization translation, *chart question answering* (with applications in accessiblity of charts), automated *data exploration* and *automated data stories*.

To the best of our knowledge, LIDA is the first tool to formulate visualization/infographic generation as a multi-step generation task and demonstrate an end-to-end pipeline that addresses a variety of subtasks.

2 Related Work

LIDA is informed by research on large foundation models applied to creative tasks across modalities such as text and images, as well as advances in automated generation of visualizations and infographics.

2.1 Foundation Models for Creative Tasks

Advances in large transformer-based (Vaswani et al., 2017) models trained on massive amounts of data (terabytes of text and images) have led to a paradigm shift where a single model demonstrates state of the art task performance across multiple data modalities such as text, images, audio and video. These models, also known as foundation models (Bommasani et al., 2021), have been shown to be effective for a variety of *human creativity* tasks. LLMs like the GPT3 series (Brown et al., 2020), OPT (Zhang et al., 2022), PALM (Chowdhery et al., 2022), LAMBDA (Cohen et al., 2022) learn complex semantics of language allowing them to be effective in tasks such as text summarization, question answering. Code LLMs such as Codex (Chen et al., 2021), AlphaCode (Li et al., 2022), InCoder (Fried et al., 2022) show state of the art performance on a suite of code intelligence tasks. Finally, models such as CLIP (Radford et al., 2021), DALLE (Ramesh et al., 2022, 2021) and Stable Diffusion (Rombach et al., 2022) have shown state of the art capabilities on image generation tasks such as image captioning, image editing, and image generation.

In this work, we adopt insights from Program-Aided Language models (Gao et al., 2022) - a setup where LLMs read natural language problems and generate programs as the intermediate reasoning steps, but offload the solution step to a runtime such as a python interpreter. Specifically we leverage the language modeling capabilities of LLMs in generating visualization goals, and their code writing capabilities in generating visualization code. The output of this step is then compiled and filtered using an interpreter to generate the final visualization. Finally, the image output from this process is used as an input to image generation models in generating stylized infographics.

2.2 Automated Visualization Generation

Extant research that support users in generating visualizations have explored multiple approaches such as heuristics, task decomposition or learning based approaches. Heuristics based approaches explore properties of data in generating a search space of potential visualizations (Wongsuphasawat et al., 2017), ranking these visualizations based on quality attributes(Luo et al., 2018; Moritz et al., 2018) and presenting them to the user. For example, DeepEye (Luo et al., 2018) enumerates all possible visualizations and classifies/ranks them as "good" or "bad" using a binary decision tree classifier while Voyager (Wongsuphasawat et al., 2017) uses heuristics to enumerate the space of visualizations. The primary limitation here is that heuristics can be tedious to maintain, may have poor coverage of the visualization space and does not leverage information encoded in existing datasets. More recent work has explored a task decomposition approach where the AUTOVIZ process is decomposed into multiple tasks that are solved individually and aggregated to yield visualizations. For example NL4DV (Narechania et al., 2020) implements a custom

[3]Reach out for early access.

query engine that parses natural language queries, identifies attributes/tasks and generates Vega-Lite specifications. A limitation of task decomposition approaches is that they are bottlenecked by the implementation performance for each step (e.g., limitations with models for disambiguating natural language queries as seen in NL4DV (Narechania et al., 2020)). Finally, learning-based approaches seek to automatically learn mappings from data in generating visualizations. For example, Data2Vis (Dibia and Demiralp, 2019) uses a sequence to sequence model that implicitly addresses AUTOVIZ subtasks by learning a mapping from raw JSON data sampled from datasets to Vega-Lite (Satyanarayan et al., 2017) specifications. Some limitations of current learning approaches is that they are limited to a single grammar, require custom models, custom paired training data and training objectives (Dibia and Demiralp, 2019; Luo et al., 2018) for each supported grammar, and do not provide a path to generating infographics. Furthermore, they do not provide mechanisms for fine-grained control of visualization output or provide robust error detection and recovery strategies.

LIDA addresses these limitations in several ways: (i) Leverages patterns learned by LLMs from massive language and code dataset, applying this knowledge to subtasks. (ii) Provides a single grammar-agnostic pipeline that generates visualization in multiple programming languages and visualization grammars. (iii) Supports natural language based control of generated visualizations (iv) leverage emergent capabilities of large language models such chain of thought reasoning (Kojima et al., 2022; Wei et al., 2022; Shi et al., 2022a) improve reliability of generated visualizations and model calibration (Kadavath et al., 2022) (predictions on correctness probabilities of visualizations) as well as self consistency (Wang et al., 2022) in ranking results. LIDA supports a fully automatic mode where an LLM is used to discover meaningful goals/hypotheses (fields to visualize, questions to ask) or a semi automatic mode where the user provides a hypothesis and it generates a visualization. Finally, it provides a mechanism for generating infographics that are data-faithful and aesthetically pleasing. By choosing to cast visualization/infographic generation as generation tasks that offloads core problem solving to LLMs and IGMs, LIDA simplifies the design and maintenance of such systems.

2.3 Infographics Generation

Infographics (information graphics) are visual artifacts that seek to convey complex data-driven narratives using visual imagery and embellishments (Harrison et al., 2015). Existing research has shown that infographics are aesthetically pleasing, engaging and more memorable (Tyagi et al., 2021; Harrison et al., 2015; Haroz et al., 2015), at no additional cost to the user (Haroz et al., 2015). These properties have driven their applications in domains like fashion, advertisemnt, business and general communications. However, the creation of infographics that convey data insights can be a tedious process for content creators, often requiring skills across multiple tools and domains. Research on infographic generation have mainly explored the creation of pictographs (Haroz et al., 2015) - replacing the marks on traditional charts with generated images and learning to extract/transfer styles from existing pictographs (Shi et al., 2022b). In this work, we extend this domain to exploring the generation of both visual marks as well as generating the entire chart based on natural language style descriptions using large image generation models such as DALLE (Ramesh et al., 2022, 2021) and Stable Diffusion (Rombach et al., 2022). This approach also enables user-generated visual styles and personalization of visualizations to fit user preferences such as color palettes, visual styles, fonts etc.

3 The LIDA System

LIDA comprises of 4 core modules - a SUMMARIZER, a GOAL EXPLORER, a VISGENERATOR and an INFOGRAPHER (see Fig 1). It also provides a USER INTERFACE with affordances for data upload, summary enrichment, conversational NL interface for visualization/infographic refinement.

3.1 SUMMARIZER

LLMs are capable zero shot predictors, able to solve multiple tasks with little or no guiding examples. However, they can suffer from hallucination e.g., generating text that is not grounded in training data or the current task. One way to address this is to *augment* (Mialon et al., 2023) the LLM with grounding context. Thus, the goal of the summarizer is to produce an information dense but compact (i.e., that maximizes the limited context token budget of these models) NL summary for a given dataset that is *useful as grounding context* visualization tasks. A useful context is defined as one that *contains*

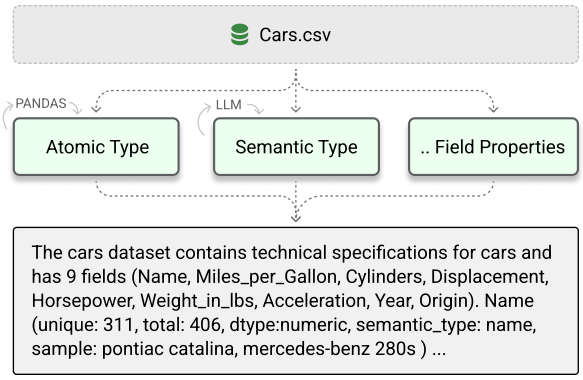

Figure 3: The SUMMARIZER module constructs a NL summary of data from extracted types (atomic and semantic) and other field properties.

information an analyst would need to understand the dataset and the tasks that can be performed on it such as semantic types (Zhang et al., 2019) and other data properties. Thus, the generated summary is based on the following dataset properties (see Fig 3) - (i) Extracted atomic types (e.g., integer, string, boolean) based on the pandas library (McKinney, 2010) and semantic types (e.g., location, company, person) predicted using an LLM. (ii) General data field properties (e.g., # of unique samples, max and min, range etc.) and an illustrative non-null list of n samples from each column. This summary may be optionally enriched by an LLM or a user via the LIDA ui to include semantic description of the dataset (e.g., a dataset on the technical specification of cars) and each field (e.g., miles per gallon for each car).

3.2 GOAL EXPLORER

This module generates visualization goals given the *dataset context* generated by the SUMMARIZER (e.g., what is the relationship between miles per gallon and horsepower?, see Fig 1). We structure this task as a two-step decomposed LLM text generation flow given the data context - i.) ***enumerate*** NL goals (e.g., Q: what are 5 questions an experienced data scientist may ask given the data and what is the rationale? .. A: understand trends in miles per gallon over time) ii.) ***ground*** each goal as a visualization task using fields in the dataset fields (e.g. Q: Generate a visualization that tells us about trends in miles per gallon over time? A: Plot of miles per gallon per year). The first step leverages patterns learned by the LLM's of human behavior (e.g., descriptions of analysts across multiple settings) while the second step leverages patterns learned on visualization tasks.

VISGENERATOR

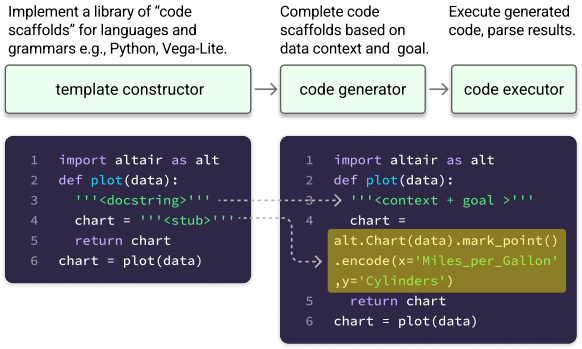

Figure 4: The VISGENERATOR module constructs visualization code scaffolds, fills a constrained section ($< stub >$) and executes the scaffold.

The VISGENERATOR generates visualization specifications based on goals produced by either the GOAL EXPLORER module or goals directly provided by a user. It is comprised of 3 submodules - a *template constructor*, a *code generator* and a *code executor*.

The *template constructor* implements a library of code scaffolds that correspond to programming languages and visualization grammars. For example, python templates support grammars such as Matplotlib, GGPlot, Plotly, Altair, Seaborn, and Bokeh. Each scaffold is an *executable program* that i.) imports relevant dependencies ii.) defines an initially empty function stub which returns a visualization specification (see Fig 4a).

At runtime, the *code generator* takes in a template, a dataset context, and a visualization goal. The documentation section of the function stub in the template is replaced with a concatenation of the dataset context and the visualization goal. A code generation LLM (applied in *fill-in-the-middle* mode (Bavarian et al., 2022)) is then used to generate n function body candidates.

Finally, the *code executor* executes[4] the completed templates and filters the results. LIDA implements several filtering mechanisms to detect errors, each with latency tradeoffs. The first is a simple filter that generates a large sample for n and removes code snippets that do not compile. The second leverages self consistency (Wang et al., 2022) in LLMs where multiple candidates are generated and the solution with the highest consensus is selected.

[4]Execution in a sandbox environment is recommended.

The third approach uses the LLM to generate a probability of correctness (Kadavath et al., 2022) for all n candidates and reranks the list based on the predicted probabilities. Note that the last two approaches are computationally expensive (require multiple forward passes through an LLM) and are not suitable for real time applications. The final output is a list of visualization specifications (and associated raster images).

3.3 INFOGRAPHER

This module is tasked with generating stylized graphics based on output from the VISGENERATOR module (see Fig 2). It implements a library of visual styles described in NL that are applied directly to visualization images. Note that the style library is editable by the user. These styles are applied in generating infographics using the text-conditioned image-to-image generation capabilities of diffusion models (Rombach et al., 2022), implemented using the *Peacasso* library api (Dibia, 2022). An optional post processing step is then applied to improve the resulting image (e.g., replace axis with correct values from visualization, removing grid lines, and sharpening edges).

3.4 USER INTERFACE

LIDA implements a user interface that communicates with the core modules over a REST and Websocket api. The user interface implements 2 main views - a data summary view and visualization view.

Data Summary View

This view allows the user to upload a dataset and explore a sample of rows in the dataset via a table view. A data upload event also triggers a call to the SUMMARIZER and GOAL EXPLORER module and displays a summary of the dataset and a list of potential goals and associated visualizations. This view also allows the user to optionally annotate and refine (e.g., add dataset or field descriptions) to the automated summary produced by the SUMMARIZER. The user may also include or exclude fields from the dataset.

Visualization View

This view allows the user to optionally provide a visualization goal (e.g., "what is the fuel efficiency per country?") or select a generated goal and then displays a generated visualization. For each visualization, intermediate output from the models (underlying data summary, visualization specification, code scaffold) are shown as explanations to aid in sensemaking, and debugging. Furthermore, the user is able to *interactively update* the generated visualizations based on NL descriptions of intent (e.g. "*change the x-axis to horsepower*", "*show only cars with miles per gallon < 35*", "*translate the chart title to spanish|hindi|maltese ..*", "*convertir esto en un gráfico de barras*"). Note that the NL interface inherits the multilingual language capabilities of the underlying LLM, enabling users to interact with the system in their native language.

4 Design Reflections

Building a system that leverages foundation models (text and images) involves engineering decisions across a wide design space. In this section, we briefly reflect on some of the design choices we made for LIDA components and the tradeoffs we considered.

4.1 Prompt Engineering

We explored multiple approaches to building prompts that maximized the probability of the LLM solving each subtask.

Prompt Design: (i) SUMMARIZER: We found that improving the richness of the summary (qualitative NL description, including semantic types) was ***critical*** to improved quality of generated goals and visualization code. Implementation wise, we began with a manually crafted summary of the data (see Section 3.1), and then enrich it via calls to an LLM *and* optional user refinement of the summary. (ii) GOAL EXPLORER: Providing instructions where fields and rationale are linked via symbols (e.g., plot a histogram of field X vs Y to show relationship between X and Y) nudges the model to use exact dataset field names, and minimizes the occurrence of hallucinated fields. Prompt engineering also provides mechanisms to bake in visualization best practices e.g. *avoid pie charts*, *apply visualization best practices*, *Imagine you are a highly experienced visualization specialist and data analyst*. (iii) VISGENERATOR: Casting visualization code generation as a *fill-in-the-middle* problem (as opposed to completion) ensures the model generates executable code *focused* on the task. For example, in Fig 4, the model is *instructed* to generate only the $< stub >$ portion of the code scaffold. (iv) Overall, we found that setting a low temperature ($t = 0$; generating the most likely visualization) coupled

with a per-grammer code scaffold provided the best results in terms of yielding code that correctly compiles into visualization specifications and faithfully addresses the subtask. We also explored prompt formulations that addressed multiple tasks to minimize costs (e.g., we asked the LLM to describe the dataset *and* each column in the dataset in a single call).

4.2 Infographic Generation

We found that setting a low *strength* parameter ($0.25 < strength < 0.45$) for the stable diffusion model and using parsimonious style prompts resulted in stylized images that were faithful to the general *structure* of the original visualization, minimizing distorted or irrelevant imagery. This sort of controlled generation is *necessary* to avoid the distraction (Haroz et al., 2015) that can arise from superfluous imagery in infographics.

4.3 Natural Language Interaction

(i) HYBRID INTERFACE: Providing a hybrid interface that allows traditional direct manipulation steps in creating visualizations (e.g., selecting which fields to use), paired with a NL interface allows users to leverage existing mental models with traditional visualization tools as well as the NL affordances of LIDA. (ii) NL INTERACTION MODES: We provide two NL interaction models - *generation* and *refinement*. The first explores initially generating a visualization based on a described high level goal. The second mode follows insights from (Mitra et al., 2022) and allows users to refine (apply operations on) an existing visualization via a conversational interface that tracks conversation history.

5 Limitations

While LIDA demonstrates clear advances in how we can support users in authoring visualizations and infographics, there are several limitations that offer a natural avenue for future research.

Low Resource Grammars: The problem formulation introduced in LIDA depends on the underlying LLMs having *some* knowledge of visualization grammars as represented in *text and code* in its training dataset (e.g., examples of Altair, Vega, Vega-Lite, GGPLot, Matplotlib, *represented in* Github, Stackoverflow, etc.). For visualization grammars not well represented in these datasets (e.g., tools like Tableau, PowerBI, etc., that have graphical user interfaces as opposed to code representations), the performance of LIDA may be limited without additional model fine-tuning or translation. Furthermore, performance may be limited for complex tasks (e.g., tasks requiring complex data transformations) beyond the expressive capabilities of specific grammars. Further research is needed to: i.) study effects of strategies like task disambiguation ii.) impact of task complexity and choice of programing language/grammar on performance.

Deployment and Latency: Large language models (e.g., GPT3 used in this work) are computationally expensive and require significant compute resources to deploy at low latency. These costs can prove to be impractical for *real-world application*. In addition, the current setup includes a code execution step which is valuable for verification but increases deployment complexity (requires a sandbox). Thus, there is opportunity to: i.) train smaller capable LLMs (Touvron et al., 2023) finetuned on a curated dataset of programming languages and visualization grammars .ii) design vulnerability mitigation approaches such as limiting program scope or generating only input parameters for visualization grammar compilers.

Explaining System Behavior: The approach discussed in this paper simplifies the design of visualization authoring systems, but also inherits interpretability challenges associated with large language models. While LIDA offers intermediate outputs of the model (e.g., generated code and specifications) as *explanations*, there is a need for additionl research in explaining system behavior (conditions when they are needed) and providing feedback to the user.

System Evaluation: Benchmarking LLM's on creativity tasks can be challenging. The current study does not cover a comprehensive benchmark of LIDA's performance on a variety of datasets and visualization grammars or specifically evaluate the *quality* of the generated visualizations. Given the general absence of visualization generation tasks in LLM benchmarks (Liang et al., 2022), there are research opportunities to i.) introduce or adapt existing evaluation metrics and datasets for automated visualization subtasks. For example, Data2Vis (Dibia and Demiralp, 2019) introduced metrics to evaluate language and grammar syntax correctness for generated visualizations. ii.) qualitative studies that evaluate the model behavior such as failure cases, native ability to encode visualization best

practices etc. iii.) study the impact of tools like LIDA on user *creativity* while authoring visualizations.

6 Conclusion

In this work, we formulate the visualization generation task as a multi-stage text (and code) generation problem that can be addressed using large language models. We present LIDA - a tool for the automatic generation of grammar-agnostic visual**li**zations an**d** infogr**a**phics. LIDA addresses limitations of current automatic visualization systems - automatic generation of hypothesis/goals given datasets, conversational interface for controllable visualization generation and refinement, support for multiple visualization grammars using the same pipeline and the ability to generate infographics. LIDA is effective compared to state of the art systems; it offers a simplified system implementation and leverages the immense language modeling and code generation capabilities of LLMs in implicitly solving complex visualization subtasks. We hope modules implemented in LIDA will serve as useful building blocks in enabling complex creative workflows such as *visualization translation*, *chart question answering*(with applications in accessiblity of charts), automated *data exploration* and *automated storytelling*.

Acknowledgements

This manuscript has benefited from comments and discusions with members of the HAX group (Saleema Amershi, Adam Fourney, Gagan Bansal), VIDA group (Steven Drucker, Bongshing Lee, Dan Marshall), Rick Barraza and others at Microsoft Research.

References

Scott Bateman, Regan L Mandryk, Carl Gutwin, Aaron Genest, David McDine, and Christopher Brooks. 2010. Useful junk? the effects of visual embellishment on comprehension and memorability of charts. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 2573–2582.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Aaron Daniel Cohen, Adam Roberts, Alejandra Molina, Alena Butryna, Alicia Jin, Apoorv Kulshreshtha, Ben Hutchinson, Ben Zevenbergen, Blaise Hilary Aguera-Arcas, Chung ching Chang, Claire Cui, Cosmo Du, Daniel De Freitas Adiwardana, Dehao Chen, Dmitry (Dima) Lepikhin, Ed H. Chi, Erin Hoffman-John, Heng-Tze Cheng, Hongrae Lee, Igor Krivokon, James Qin, Jamie Hall, Joe Fenton, Johnny Soraker, Kathy Meier-Hellstern, Kristen Olson, Lora Mois Aroyo, Maarten Paul Bosma, Marc Joseph Pickett, Marcelo Amorim Menegali, Marian Croak, Mark Díaz, Matthew Lamm, Maxim Krikun, Meredith Ringel Morris, Noam Shazeer, Quoc V. Le, Rachel Bernstein, Ravi Rajakumar, Ray Kurzweil, Romal Thoppilan, Steven Zheng, Taylor Bos, Toju Duke, Tulsee Doshi, Vincent Y. Zhao, Vinodkumar Prabhakaran, Will Rusch, YaGuang Li, Yanping Huang, Yanqi Zhou, Yuanzhong Xu, and Zhifeng Chen. 2022. Lamda: Language models for dialog applications. In *arXiv*.

Victor Dibia. 2022. Interaction design for systems that integrate image generation models: A case study with peacasso.

Victor Dibia and Çağatay Demiralp. 2019. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications*, 39(5):33–46.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. 2022. Incoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Gra-

ham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Steve Haroz, Robert Kosara, and Steven L Franconeri. 2015. Isotype visualization: Working memory, performance, and engagement with pictographs. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 1191–1200.

Lane Harrison, Katharina Reinecke, and Remco Chang. 2015. Infographic aesthetics: Designing for the first impression. In *Proceedings of the 33rd Annual ACM conference on human factors in computing systems*, pages 1187–1190.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018. Deepeye: Creating good data visualizations by keyword search. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD, pages 1733–1736.

Wes McKinney. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Rishab Mitra, Arpit Narechania, Alex Endert, and John Stasko. 2022. Facilitating conversational interaction in natural language interfaces for visualization. In *2022 IEEE Visualization and Visual Analytics (VIS)*, pages 6–10. IEEE.

Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. 2018. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE transactions on visualization and computer graphics*, 25(1):438–448.

Arpit Narechania, Arjun Srinivasan, and John Stasko. 2020. Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):369–379.

Luca Podo, Bardh Prenkaj, and Paola Velardi. 2023. Machine learning for visualization recommendation systems: Open challenges and future directions. *arXiv preprint arXiv:2302.00569*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. 2022 ieee. In *CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685.

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-lite: A grammar of interactive graphics. *IEEE TVCG (Proc. InfoVis)*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. 2022a. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*.

Yang Shi, Pei Liu, Siji Chen, Mengdi Sun, and Nan Cao. 2022b. Supporting expressive and faithful pictorial visualization design with visual style transfer. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):236–246.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Anjul Tyagi, Jian Zhao, Pushkar Patel, Swasti Khurana, and Klaus Mueller. 2021. User-centric semi-automated infographics authoring and recommendation. *arXiv preprint arXiv:2108.11914*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting visual analysis with partial view specifications. In *ACM CHI*.

Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, and Wang-Chiew Tan. 2019. Sato: Contextual semantic type detection in tables. *arXiv preprint arXiv:1911.06311*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

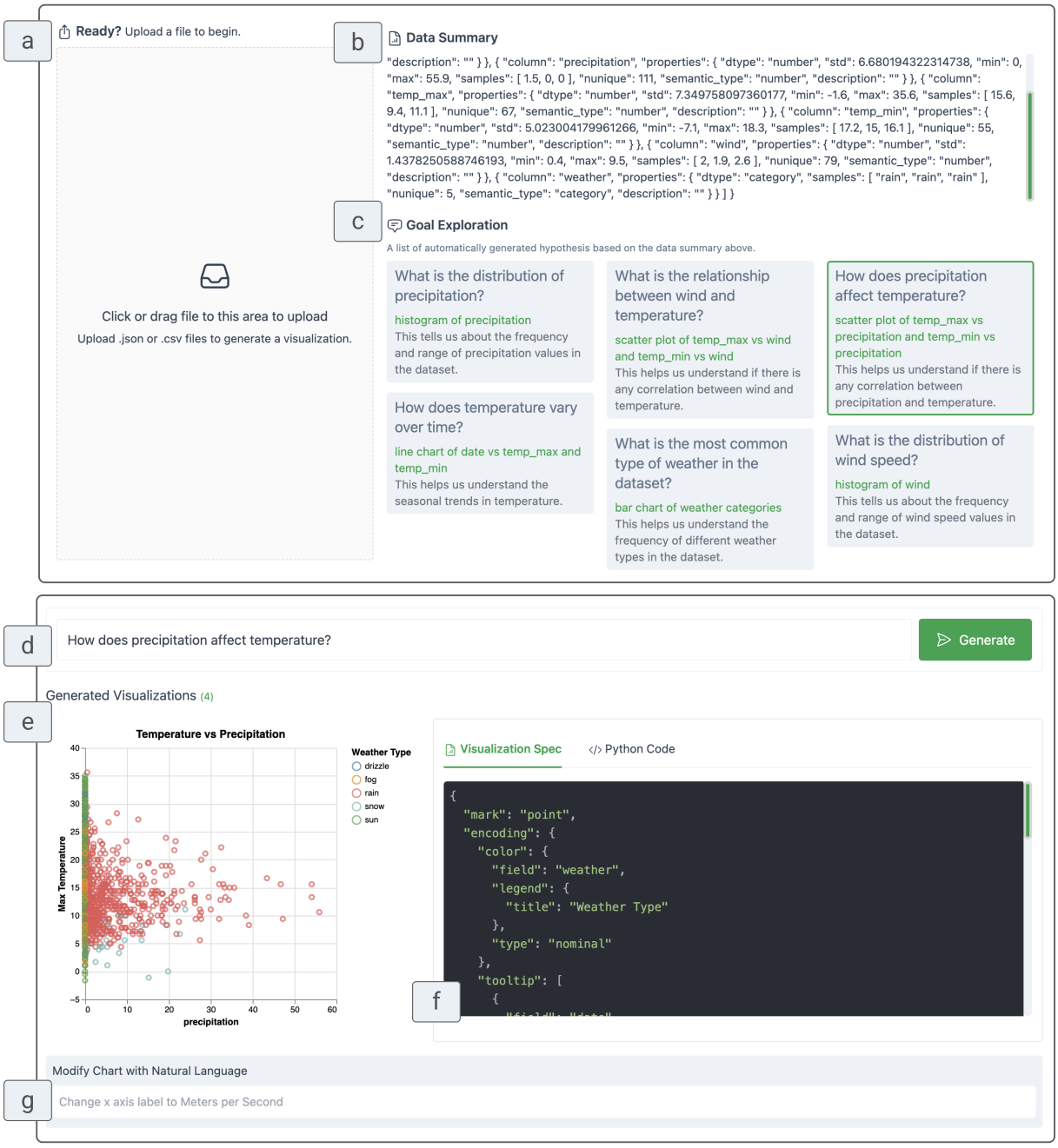

Figure 5: The LIDA USER INTERFACE. a.) The user can upload a dataset in multiple formats (csv, json, excel). b.) Data summary view displays a natural language summary of the dataset with affordances to enrich the representation. c.) A list of generated goals and rationale related to the data. d.) User goals specified in natural language (multilingual support) e.) Generated visualization f.) View of specification and python code used to create generation g.) Interactive chat-style window for refining an existing visualization in natural language.

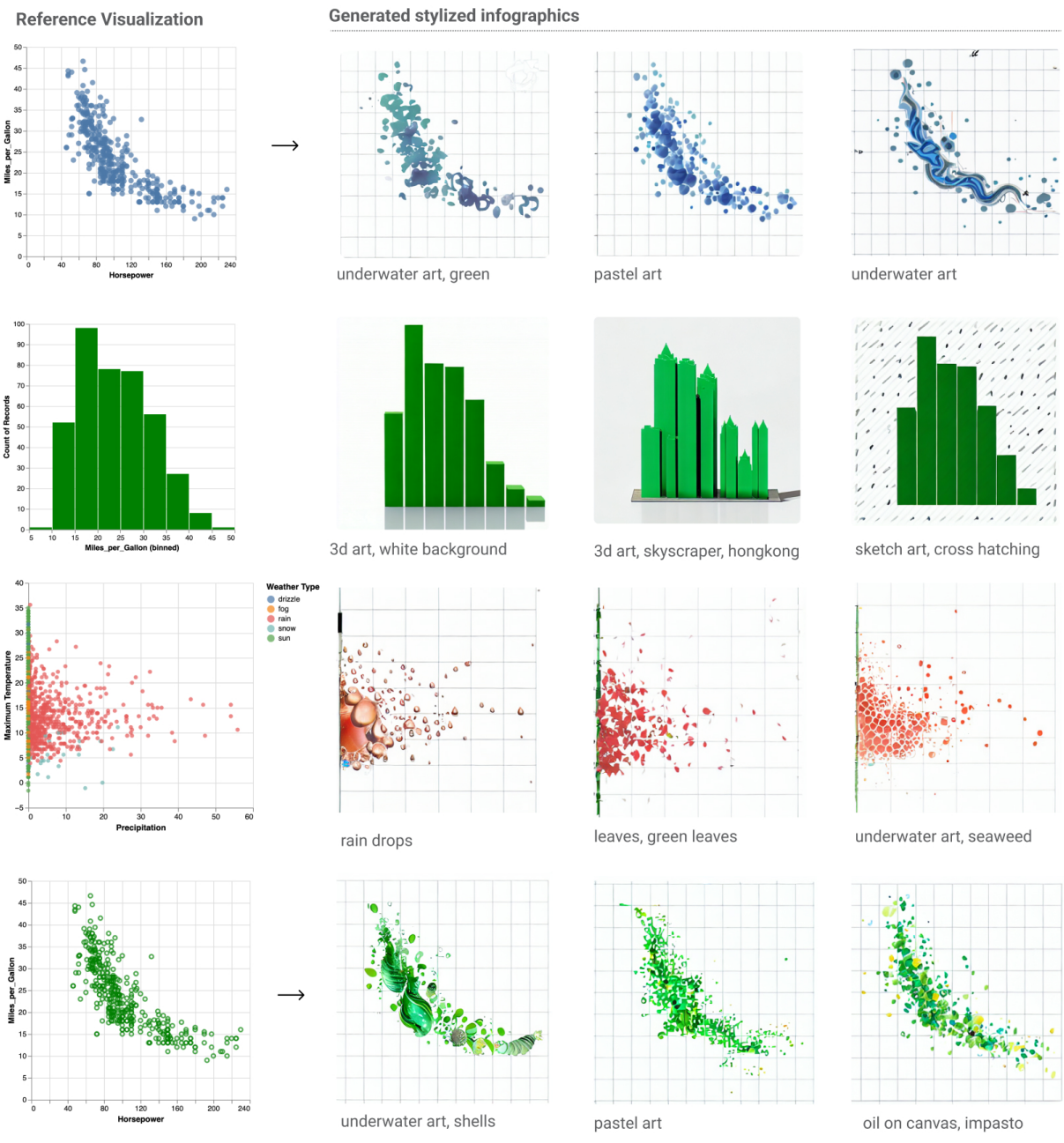

Figure 6: Examples of infographics generated with LIDA.

Rule-based summary

```
{'name': '',
 'dataset_description': '',

 'columns': ['Name',
  'Miles_per_Gallon',
  'Cylinders',
  'Displacement',
  'Horsepower',
  'Weight_in_lbs',
  'Acceleration',
  'Year',
  'Origin'],
 'properties': [{'column': 'Name',
   'properties': {'dtype': 'string',
    'samples': ['ford gran torino (sw)', 'volvo 264gl',
'ford country'],
    'nunique': 311,
    'min': 'amc ambassador brougham',
    'max': 'vw rabbit custom',
    'semantic_type': ''}},
  {'column': 'Miles_per_Gallon',
   'properties': {'dtype': 'number',
    'std': 7.8159843125657815,
    'samples': [33.0, 15.0, 27.0],
    'nunique': 129,
    'min': 9.0,
    'max': 46.6,
    'semantic_type': ''}},
  {'column': 'Cylinders',
   'properties': {'dtype': 'number',
    'std': 1.7121596315485297,
    'samples': [6, 4, 6],
    'nunique': 5,
    'min': 3,
    'max': 8,
    'semantic_type': ''}},
  ....
  {'column': 'Origin',
   'properties': {'dtype': 'category',
    'samples': ['USA', 'USA', 'USA'],
    'nunique': 3,
    'min': 'Europe',
    'max': 'USA',
    'semantic_type': ''}}]
    }
```

Enriched summary using an LLM

```
{'name': 'Car Dataset',
 'dataset_description': 'A dataset containing information
about cars',
 'columns': ['Name',
  'Miles_per_Gallon',
  'Cylinders',
  'Displacement',
  'Horsepower',
  'Weight_in_lbs',
  'Acceleration',
  'Year',
  'Origin'],
 'properties': [{'column': 'Name',
   'properties': {'dtype': 'string',
    'samples': ['ford gran torino (sw)', 'volvo 264gl',
'ford country'],
    'nunique': 311,
    'min': 'amc ambassador brougham',
    'max': 'vw rabbit custom',
    'semantic_type': 'vehicle_model'}},
  {'column': 'Miles_per_Gallon',
   'properties': {'dtype': 'number',
    'std': 7.8159843125657815,
    'samples': [33.0, 15.0, 27.0],
    'nunique': 129,
    'min': 9.0,
    'max': 46.6,
    'semantic_type': 'fuel_efficiency'}},
  {'column': 'Cylinders',
   'properties': {'dtype': 'number',
    'std': 1.7121596315485297,
    'samples': [6, 4, 6],
    'nunique': 5,
    'min': 3,
    'max': 8,
    'semantic_type': 'engine_cylinders'}},
  ....
  {'column': 'Origin',
   'properties': {'dtype': 'category',
    'samples': ['USA', 'USA', 'USA'],
    'nunique': 3,
    'min': 'Europe',
    'max': 'USA',
    'semantic_type': 'vehicle_origin'}}]
    }
```

Figure 7: To create a summary of the data, we first generate a rule based summary using the PANDAS library to extract atomic *dtypes* and general column properties. Next, we apply an LLM to enrich this summary by adding a description of the dataset and columns given the current features.