VIST5: An Adaptive, Retrieval-Augmented Language Model for Visualization-oriented Dialog

Henrik Voigt[1], **Nuno Carvalhais**[2], **Monique Meuschke**[3], **Markus Reichstein**[2], **Sina Zarrieß**[4], **Kai Lawonn**[1]
[1]University of Jena [2]MPI Biogeochemistry [3]University of Magdeburg [4]Bielefeld University
[1]first.last@uni-jena.de, [2]first.last.bgc-jena.mpg.de
[3]last@isg.cs.uni-magdeburg.de, [4]first.last@uni-bielefeld.de

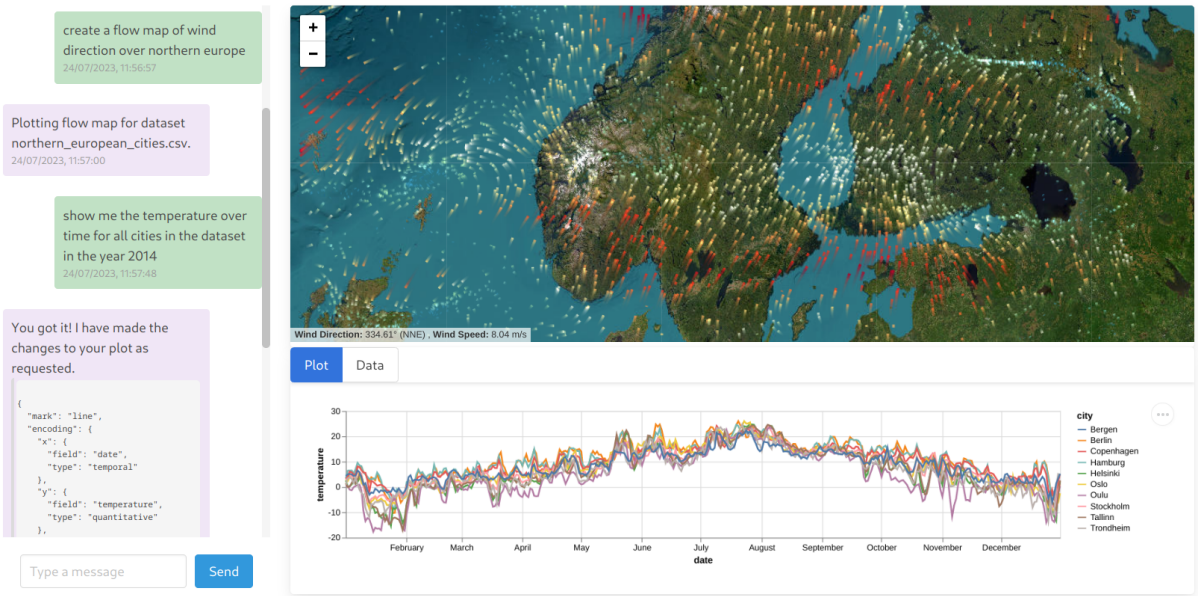


Figure 1: VIST5 makes it easy for researchers and professionals to explore their data using natural language. Users articulate their visualization preferences in a **chat window**, displayed in the left column. The panel lists the responses of the dialog agent, containing both text and custom Vega-Lite visualization code. The right column contains two visualization tools that can be controlled from the chat. At the top, a **geographical map** displays geo-related plots, such as flow visualizations of wind directions. Below is a display area for **Vega-Lite** visualizations that are generated based on user queries to the dataset.

Abstract

The advent of large language models has brought about new ways of interacting with data intuitively via natural language. In recent years, a variety of visualization systems have explored the use of natural language to create and modify visualizations through visualization-oriented dialog. However, the majority of these systems rely on tailored dialog agents to analyze domain-specific data and operate domain-specific visualization tools and libraries. This is a major challenge when trying to transfer functionalities between dialog interfaces of different visualization applications. To address this issue, we propose VIST5, a visualization-oriented dialog system that focuses on easy adaptability to an application domain as well as easy transferability of language-controllable visualization library functions between applications. Its architecture is based on a retrieval-augmented T5 language model that leverages few-shot learning capabilities to enable a rapid adaptation of the system.

1 Introduction

The field of visualization has witnessed a surge of interest in integrating dialogue interfaces into visualization applications, leading to the development of various visualization-oriented natural language interfaces (V-NLI) (Narechania et al., 2020; Luo et al., 2021b; Liu et al., 2021; Kim et al., 2021). The goal of these systems is to generate visualizations from natural language queries and modify them accordingly in interaction with the user. However, visualization applications exist in various domain contexts, which require specific vocabulary to be parsed and mapped to custom functionalities.

Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 70–81
December 6-10, 2023

For example, a visualization application that helps researchers analyze climate data will handle different user intent and different, domain-specific visualization libraries than an application in a medical context (Srinivasan et al., 2021; Gao et al., 2015). Certain types of visualization techniques, such as *bar charts*, *line charts*, or *scatter plots*, are very general, so they can be used in almost any domain. Others, such as *flow maps* for the visualization of wind vectors, are not and their access via the natural language interface must be integrated with great effort. Transferring a set of solutions, such as successfully mapping user queries to visualization library functions, from one V-NLI to another without writing new code is still a challenging task. It would be ideal if created functionality could be transferred between V-NLI applications by simply showing the system how to use a particular library with a few examples.

In this paper, we introduce VIST5, a V-NLI that helps users perform text-related visualization tasks while being adaptive to the visualization libraries of the application domain. The system implements a retrieval-augmented language model trained on a mixture of visualization-specific text generation tasks and a large collection of general text-to-text translation tasks. Its retrieval augmentation allows modular extension with domain-specific user commands and portability of functionality between applications. Moreover, the language model meets the requirements of small model size, fast trainability, and fast inference on commodity hardware. We illustrate the adaptation to the specifics of a domain using the example of **climate data exploration**.

Our contributions can be summarized as follows:

- **Efficient Multi-Task Architecture.** Introduction of an efficient and generic multi-task architecture for text-related visualization tasks.
- **Retrieval-Augmented Dialogue System.** The presentation of a dialog system that uses an *information retrieval* component to ground the dialog in knowledge retrieval from external resources. This allows a smaller model size while exploiting knowledge from external databases.
- **Modular Extensibility via Few-Shot Paradigm.** Leveraging the *few-shot* capabilities of the language model to enable modular extensibility and portability of user intents between applications, as well as integration of new custom intents in minutes.

For a demo video of the VIST5 system please visit **https://youtu.be/bsgaV7hjlGs**.

2 Related Work

Natural language interfaces for data visualization have recently emerged as a powerful combination of visualization and NLP techniques. In their comprehensive survey, Shen et al. (2021) provide an overview of how natural language interaction can be integrated into the visualization pipeline of Card (1999). Voigt et al. (2021, 2022) elaborate on the different visualization tasks that can be facilitated by natural language interactions. The resulting **V-NLI pipeline** is shown in Figure 2. The following is a sequential listing of the steps in the V-NLI pipeline paired with recent work in each step.

Query Interpretation. Interpreting the query is about identifying the subset of the *data* the user wants to see and the *actions* the user wants to perform on the data. Setlur et al. (2016) introduced Eviza, which leverages a probabilistic grammar defining a rule-based interaction schema on how to react to specific types of queries. Flowsense (Yu and Silva, 2019), another rule-based semantic parsing approach, matches special utterances and maps them to visualizations in a data flow architecture. Other works focus on resolving linguistic ambiguity and vagueness in expressions using sentiment analysis and word co-occurrence (Hearst et al., 2019; Setlur et al., 2019). Recent systems have introduced neural sequence-to-sequence approaches that translate queries directly into visualizations (Luo et al., 2021b). Maddigan and Susnjak (2023) have conducted an investigation on diverse prompt designs for ChatGPT (Ouyang et al., 2022), OpenAI Codex (Chen et al., 2021), and GPT-3 (Brown et al., 2020), demonstrating the remarkable capability of these LLMs in producing high-fidelity visualizations from natural language input. Our work takes a different approach, considering that training and inferring such large models can be expensive and hardware-intensive, making them unsuitable for computationally constrained use cases. Instead, we concentrate on open access, extensibility, and modularity, offering an alternative perspective.

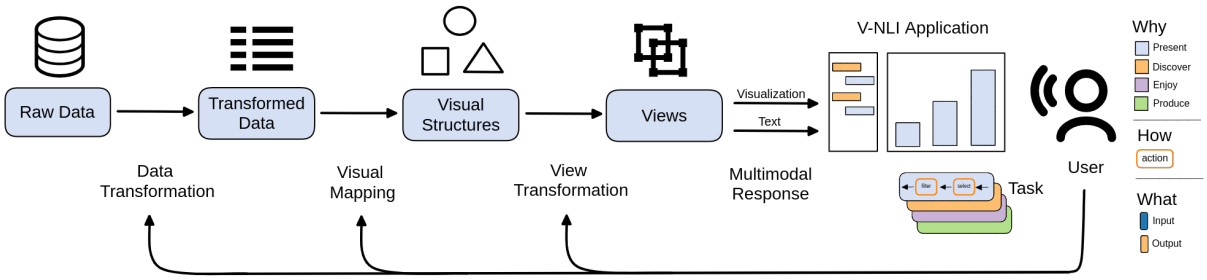


Figure 2: **V-NLI Pipeline.** Given a *user query*, the data is first *transformed*, then *mapped* to visual structures, and then *displayed* in a view. The user, on the other hand, uses the interface by accessing different stages of the pipeline via language to solve a visualization task *action by action*.

Data Transformation. Transforming the data according to the action specified by the user is the next step in the V-NLI pipeline (e.g. by *aggregation, filtering, binning, or grouping*). A set of approaches identifies transformation functions from visualization libraries through phrase matching (Gao et al., 2015; Hoque et al., 2017; Sun et al., 2010; Srinivasan and Stasko, 2017; Dhamdhere et al., 2017), others make use of a common data interface such as SQL (Zhong et al., 2017; Wang et al., 2019; Scholak et al., 2021; Xie et al., 2022; Qi et al., 2022).

Visual Mapping. In V-NLI systems, the mapping from data to visual representation is usually seen in one of two flavors: 1) the data transformation (e.g. selection of table, column, conditions) and the generation of the visualization specification (e.g. chart type, color) are integrated, as in ncnet (Luo et al., 2021b), or 2) the data transformation and visualization specification are separated, with an appropriate visualization for the resulting data being suggested after the query is executed (Wongsuphasawat et al., 2015, 2016; Zhu et al., 2020; Luo et al., 2018). Quda (Fu et al., 2020) and ADVISor (Liu et al., 2021) use neural intent classification methods that are more flexible for integrating custom visualization library functions, but still have the problem of being difficult to extend and adapt to new user intents without retraining.

View Transformation. In current systems, manipulation of visual elements in the view is primarily enabled through other channels of multimodal interaction, such as touch and gesture (Kim et al., 2021; Srinivasan et al., 2020b), as exemplified by InChorus (Srinivasan et al., 2020a). Orko (Srinivasan and Stasko, 2017) combines written or spoken text input with touch gestures to manipulate view properties, as does Valletto (Kassel and Rohs, 2018).

3 VIST5 System

The VIST5 system is composed of a language model (Section 3.1), a dialog management component (Section 3.2) that controls the memory and API calls to the various visualization libraries used, and a user interface (Section 3.3). The system architecture and the query execution process are shown in Figure 3. The open-source code of the system is available at **https://github.com/clause-bielefeld/VIST5.git**.

3.1 Language Model

The model architecture closely aligns with T5-base and features 12 encoder and decoder blocks with a token embedding dimension of 768 (Raffel et al., 2020). We employ an input context width of 2048 tokens to match the length of the input prompt. Natural language queries are tokenized using the SentencePiece tokenizer from Kudo and Richardson (2018) based on a 32,000 subword vocabulary. In total, this results in a size of 220 million parameters. The model is quantized and deployed in an ONNX runtime, which leads to a small memory footprint of only 225 MB (ONNX Runtime developers, 2018). We initialize with pre-trained FLAN-T5-base (Chung et al., 2022) model weights, which are obtained from the huggingface model hub (Wolf et al., 2020).

Datasets. We fine-tune the language model using the following datasets:

- **nvbench.** nvbench is the largest dataset available for the NL2VIS task (Luo et al., 2021a). In nvbench, text queries are translated into Vega-Lite JSON specifications. The dataset contains a large number of 25,750 examples from 750 data tables in 105 domains.

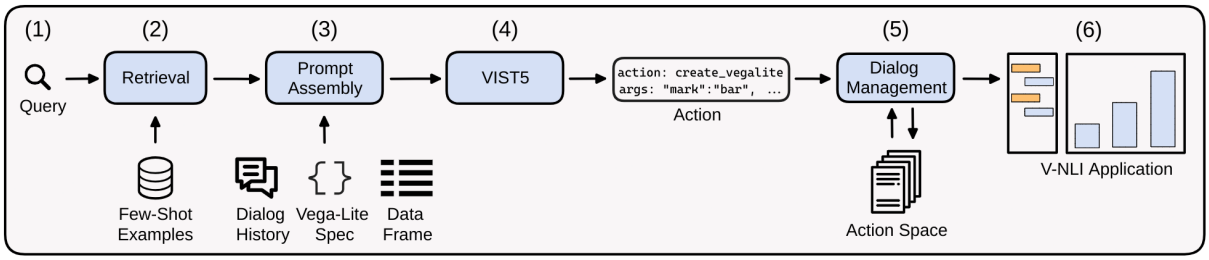


Figure 3: **VIST5 system architecture.** An example query interpretation includes the following steps: *1)* The query is tokenized and embedded into a neural embedding vector. *2)* The retrieval component returns examples relevant to the query from long-term memory. *3)* If similar examples are found, they are included in the prompt along with the *visualization state, table state* and *dialog history*. *4)* The prompt is fed into the model, which predicts an *action* and *arguments* for that action. *5)* The action is validated by the dialog management component and then executed. *6)* The output of the action is passed on to the frontend, where it leads to an update of the visualization.

- **NIv2.** The natural instructions dataset is used for **few-shot instruction** fine-tuning (Wang et al., 2022). The model is trained in such a way that it first sees three similar input/output examples in the prompt before generating a response to the current query. This training objective was explicitly chosen to train the T5 model on cases where **few-shot examples** are available in addition to an input. The goal is to train it to derive a solution (e.g., how to call a particular function) based on given examples and then apply it to the input.
- **Domain-Specific Dialogs.** The VIST5 system is equipped with an online *annotation tool* to capture domain-specific utterances and commands during runtime. We employed it to collect 300 dialog turns from researchers exploring the system. This very small dataset contains contextual queries from the domain of climate science. It is used as a showcase to demonstrate how the annotation tool can be used to adapt the model to a specific domain.

From the above datasets, we use nvbench and the domain-specific dialogs in their entirety. From NIv2, we take a random sample of 50k. We then use an NVIDIA A6000 GPU to fine-tune the language model for four hours (one epoch).

3.2 Dialog Management

To manage the dialog, we use two additional components. The first is the agent's short-term memory, which stores the *status of the visualization* and the currently selected *data table* as well as the most recent *dialog history*. The second is a long-term memory, which is a vector database of domain-specific *few-shot examples*.

3.2.1 Short Term Memory

The *visualization state* in our application consists of the composition of the currently displayed Vega-Lite chart. This is a JSON object that contains all the properties of the visualization such as *mark* and *channel encodings* as well as *data transformations* like *filters* or *aggregations*. The Vega-Lite JSON object is flattened and converted to a normalized JSON string (Wes McKinney, 2010). The *table state* consists of a Pandas dataframe (pandas development team, 2020), which is serialized as the header, followed by the first three rows. The *dialog history* is stored as a sequence of query/response pairs.

3.2.2 Long Term Memory

The main task of the long-term memory is to adapt the application to the context of use, e.g., domain-specific utterances, libraries, and functions that are used during the analysis of **climate data**. This is realized by storing a list of application-specific few-shot examples. A few-shot example is an input-output pair that contains an example user input and the desired *action*, as well as the *arguments* that the model should use to execute that action. An example to call a function of a domain specific library looks like this: `INPUT: show me a heat map of temperature, OUTPUT: action: create_heat_map; args: "column": "temperature"`. During runtime, a Sentence-Transformer (Reimers and Gurevych, 2019) is used to encode the input query into a neural embedding vector. Then, the cosine similarities between the encoded query vector and all stored encoded few-shot example vectors are computed. All examples that exceed a similarity threshold α are kept. We set α to a similarity value of 0.8. This ensures that

only very relevant examples are returned. Of the retrieved examples, the top 3 are then passed into the prompt. If no example exceeds the threshold, no example is returned and the model must respond to the input without further assistance based on the knowledge contained in its weights.

3.2.3 API Orchestration

To manage the different visualization libraries used, all functionalities (= function names and their arguments in JSON format) are listed in an *action space*. The interpretation of a request from perception to final response is as follows: Upon receiving a user request, relevant examples are first retrieved from long-term memory. The prompt is then assembled from these (potentially) retrieved *few-shot examples*, the current *visualization state* as a Vega-Lite JSON string, the *table state*, and the *user input* (see Appendix A for details). Based on this prompt, the model generates an *action* and the corresponding *arguments*. After generation, the control loop checks to see if the generated action exists in the action space, and if it does, the function is called and executed with the specified arguments. The output of this function is then sent to the frontend, where it causes a change in the targeted visualization display.

3.3 User Interface

The user interface is built in HTML, CSS, and JavaScript (see Figure 1). The backend, which serves the website and hosts the language model for inference, is based on fastAPI (tiangolo, 2023).

Visualization Display. The visualization area consists of a geographic map onto which the climate data is projected. To create the map the visualization library leaflet (leaflet, 2023) is used. Below the map, a display for Vega-Lite visualizations (Satyanarayan et al., 2018) is provided. The visualization is dynamically updated with new visualization specifications generated by the language model based on user requests.

Chat Window. On the left side, there is a chat window that contains the dialog history of the conversation. It allows the user to submit requests to the system and view the exact system responses including the generated Vega-Lite specs.

Online Annotation Tool. After receiving a response, the user can interactively edit the created Vega-Lite specification if desired. If a customized Vega-Lite specification is to be used as a training example in the future, it can also be immediately submitted back to the system in this manner.

Data Display. The Vega-Lite display can be switched to a data display. It shows an overview of the selected data set with the column headers of the data frame, their data types, and the first 1k rows of the data set.

4 Features

The focus of the system is to provide visualizations in response to user queries to help users solve application-specific visualization tasks as defined by Brehmer and Munzner (2013). In the VIST5 system, this involves three main tasks: 1) translating a natural language query into a visualization specification, 2) engaging in a domain-specific analytical conversation by exchanging contextual queries to gain insight into the data, and 3) customizing a visualization specification to meet user needs. To measure the response quality of the system in these tasks, we conducted a user study with 24 participants. It revealed that the system provided high-quality responses to diverse visualization requests, and that the vast majority of few-shot requests were also successful. Of particular note is that the users felt really engaged with the system, as evidenced by the high average number of user turns per dialog of 11.6. A detailed description of the study can be found in Appendix B.

4.1 Natural Language Query to Visualization

The Natural Language Query to Visualization (NL2VIS) task is the most prominent task supported by the system (Luo et al., 2021a). Given a query, the system responds with a Vega-Lite specification that it believes is the best one to help users answer their question. To demonstrate, consider the query: `"Show me Seattle's temperature in 2018 as a line chart"`. The query is entered into the dialog interface and sent to the backend. Since the model was trained on this task, there are no few-shot examples stored in long-term memory for it. As a result, no examples are added to the prompt. The prompt is then fed to the model. The model recognizes the NL2VIS request and generates a `create_vegalite` action with the appropriate arguments `"mark": "line", "encoding_x_field": "date", "encoding_x_type": "temporal", ...`. The generated specification is then converted from a normalized JSON string back to a JSON object, passed to the front end, and displayed to the user.

4.2 Analytical Conversations

Analytic conversations, consisting of a back-and-forth of contextual queries and responses, are critical because, in data exploration, no one knows where insights will be found until they see the data. Often, interest in certain aspects of the data is highly situational, leading to contextual queries. For example, a user might first query the temperature in `Seattle`, as in the previous example. After viewing the output, the user is interested in comparing this temperature curve to the city of `New York`, which is on the other side of the continent. In this context, given the initial visualization, the user might simply ask, `"Okay, now add the temperature in New York to the plot.` This request implies to the model that *1)* the user wants to keep the temperature in `Seattle` in the plot, *2)* the user wants to add the temperature in `New York` to the plot, *3)* the year of focus is 2018, and *4)* it might be better to color the curves for the two cities differently, otherwise it will be difficult for the user to compare the two. Extending a language interface from single-turn interactions, such as NL2VIS queries, to contextual queries greatly increases its flexibility, since practical use is always contextual.

Visualization Customization. Since the Vega-Lite specifications are available to the model in the prompt, users can also customize data-only visualizations by adding titles, labels, changing colors, or swapping axes on the fly. After completing their exploration, users may want to share a plot with their colleagues to discuss an interesting trend in the temperature curves for `New York` and `Seattle` that they observed during the exploration. To accomplish this, a user could give the instruction: `"Add a title to the chart that reads Seattle vs. New York Temperature 2018"`. The model will update the plot, and once received, the user can share the visualization with a colleague.

Domain-Specific Visualizations. The analysis of climate data depends heavily on the interpretation of the measurements in the context of the geographical location of a weather station. Only when the characteristics of the environment in terms of altitude, vegetation, and urbanization can be considered together with the data, reliable conclusions can be drawn. To this end, we integrate three geo-specific plot types to expand the range of options available to climatologists working with VIST5. For example, we enable *marker plots* of weather stations on the leaflet map, giving the user an overview of where weather stations are located. A second function is the generation of *heat maps*, which can be specified by naming the column in the dataset from which a heat map is to be generated. An example would be `"Show me a heat map of precipitation"`. This is an instruction that the model has never seen during training, but it can be solved by seeing a few examples. The third geospatial map we have integrated following this paradigm is *flow maps* to visualize wind directions.

Custom Functionalities. Custom functionalities are functions that are provided by the application but usually have to be integrated into the language interface by hand, otherwise, they are inaccessible without training data. Using the few-shot paradigm, we integrate a function to *export* plots and share them with colleagues. Furthermore, it is possible to change the *map type* between satellite/dark/street/hybrid, depending on the interest of the exploration scenario. Finally, it is also possible to ask the model to *update* the weather dataset with fresh data points from the Open Meteo Weather API (open meteo, 2023). When exploring climate data on maps, it is particularly helpful to use large screens. A drawback for the language interface, in this case, is that typing-based chat is very impractical, as it is annoying to switch back and forth between the keyboard and the screen. We, therefore, decided to include a number of voice locomotion interactions in the form of few-shot examples. We use a text-to-speech service based on the VOSK library (Shmyrev and other contributors, 2022). Interactions include *zoom in/out*, *move left/right/up/down*, and *navigating* to a specific location by naming it as in `"Navigate to the city of London, please."`. The map adjusts seamlessly and exploration can continue hands-free.

5 Conclusion

In this work, we have proposed VIST5, a system that demonstrates the adaptation of a V-NLI to an application domain using online annotation and few-shot learning techniques. The system performs a retrieval-augmented dialog by using the external knowledge contained in few-shot examples to generate responses to user input. This makes it fast, modular, and easily adaptable to a user-defined domain. Unlike large language models, VIST5 focuses on small model size, fast trainability, and fast inference on commodity hardware to meet the needs of applications with privacy concerns or lim-

ited computational resources. We hope that the system will inspire the community to further improve the architecture and create more applications and datasets for visualization-oriented dialogue to promote the combination of NLP and visualization techniques.

Limitations

Compared to very large models such as GPT-4, PaLM2, or ChatGPT, VIST5's capabilities are limited to a much smaller set of tasks. The model is not a general dialog agent like, e.g., ChatGPT and only works on tasks for which it has been trained, or if it is provided with sufficient few-shot examples by the retrieval mechanism. We see this limitation as a clear trade-off that the application developer has to make between the size of the model that can be used in their application and the model properties that are needed for the current application.

A second limitation we see is the collision of similar few-shot examples when the number of tasks to be integrated via the few-shot paradigm becomes very large. This can lead to the retrieval mechanism not always returning the optimal examples and thus providing the model with incorrect starting points that reduce the response quality. A possible compromise here could be to fine-tune the sentence transformer model on the large set of few-shot examples to ensure that the optimal examples are always retrieved.

A third limitation we see is the limitation of the model to generate complete visualization specifications only from the Vega-Lite visualization library. Adding functionality from other visualization libraries such as D3.js or Observable Plot is possible via the few-shot paradigm, but the longer the visualization specifications to be generated, the more error-prone the few-shot approach becomes for small models such as T5-base (e.g., large Vega-Lite specifications can contain more than a hundred properties). We see three approaches as promising directions for the future: 1) visualization specifications for general plots, e.g. bar charts, are specified in a library-independent way and can then be parsed from the general specification into the respective library, 2) methods for integrating code documentation of specific libraries into the prompt and making it usable so that even small language models can benefit from it need to be explored, 3) for large plot specifications of specific visualization libraries, training data needs to be generated either by humans or (depending on quality requirements) by larger models, e.g. GPT-4.

Ethics Statement

The nvbench and NIv2 datasets, as well as the T5 and FLAN-T5 models, are available for research and non-commercial use. We explicitly state that the intended use of our model is to assist researchers and domain experts in their data exploration procedures by allowing them to easily generate visualizations from natural language descriptions. The reliability of the generated visualizations and their one-to-one correspondence with the underlying data set must always be verified by the user of the VIST5 system. The language model generates visualizations based on the input query and the information contained in the prompt, within its capabilities. During generation, misinterpretations or misapplied data transformations may occur, leading to incorrect results. Therefore, we encourage users not to take the results generated by the model for granted, but to verify the generation process by always double-checking the specifications provided in the chat window for the generated visualizations and making sure that they make sense in the current context given the query and dataset at hand.

Acknowledgments

This work was supported by the Carl Zeiss Foundation in the context of the *"A Virtual Workshop for Digitization in the Sciences"* and *"Interactive Inference"* projects.

References

Robert Amar, James Eagan, and John Stasko. 2005. Low-level components of analytic activity in information visualization. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 111–117. IEEE.

Matthew Brehmer and Tamara Munzner. 2013. A multi-level typology of abstract visualization tasks. *IEEE transactions on visualization and computer graphics*, 19(12):2376–2385.

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.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Inigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. Multiwoz–a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *arXiv preprint arXiv:1810.00278*.

Mackinlay Card. 1999. *Readings in information visualization: using vision to think*. Morgan Kaufmann.

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*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Kedar Dhamdhere, Kevin S McCurley, Ralfi Nahmias, Mukund Sundararajan, and Qiqi Yan. 2017. Analyza: Exploring data with conversation. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pages 493–504.

Siwei Fu, Kai Xiong, Xiaodong Ge, Siliang Tang, Wei Chen, and Yingcai Wu. 2020. Quda: natural language queries for visual data analytics. *arXiv preprint arXiv:2005.03257*.

Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G Karahalios. 2015. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th annual acm symposium on user interface software & technology*, pages 489–500.

Marti Hearst, Melanie Tory, and Vidya Setlur. 2019. Toward interface defaults for vague modifiers in natural language interfaces for visual analysis. In *2019 IEEE Visualization Conference (VIS)*, pages 21–25. IEEE.

Enamul Hoque, Vidya Setlur, Melanie Tory, and Isaac Dykeman. 2017. Applying pragmatics principles for interaction with visual analytics. *IEEE transactions on visualization and computer graphics*, 24(1):309–318.

Jan-Frederik Kassel and Michael Rohs. 2018. Valletto: A multimodal interface for ubiquitous visual analytics. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–6.

Young-Ho Kim, Bongshin Lee, Arjun Srinivasan, and Eun Kyoung Choe. 2021. Data@ hand: Fostering visual exploration of personal data on smartphones leveraging speech and touch interaction. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–17.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

leaflet. 2023. Leaflet/leaflet.

Can Liu, Yun Han, Ruike Jiang, and Xiaoru Yuan. 2021. Advisor: Automatic visualization answer for natural-language question on tabular data. In *2021 IEEE 14th Pacific Visualization Symposium (PacificVis)*, pages 11–20. IEEE.

Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. Deepeye: Towards automatic data visualization. In *2018 IEEE 34th international conference on data engineering (ICDE)*, pages 101–112. IEEE.

Yuyu Luo, Jiawei Tang, and Guoliang Li. 2021a. nvbench: A large-scale synthesized dataset for cross-domain natural language to visualization task. *arXiv preprint arXiv:2112.12926*.

Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2021b. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):217–226.

Paula Maddigan and Teo Susnjak. 2023. Chat2vis: Generating data visualisations via natural language using chatgpt, codex and gpt-3 large language models. *arXiv preprint arXiv:2302.02094*.

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.

ONNX Runtime developers. 2018. ONNX Runtime. `https://onnxruntime.ai`.

open meteo. 2023. open-meteo/open-meteo.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*.

The pandas development team. 2020. pandas-dev/pandas: Pandas.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *arXiv preprint arXiv:2205.06983*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2018. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23:341–350.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*.

Vidya Setlur, Sarah E Battersby, Melanie Tory, Rich Gossweiler, and Angel X Chang. 2016. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th annual symposium on user interface software and technology*, pages 365–377.

Vidya Setlur, Melanie Tory, and Alex Djalali. 2019. Inferencing underspecified natural language utterances in visual analysis. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 40–51.

Leixian Shen, Enya Shen, Yuyu Luo, Xiaocong Yang, Xuming Hu, Xiongshuai Zhang, Zhiwei Tai, and Jianmin Wang. 2021. Towards natural language interfaces for data visualization: A survey. *arXiv preprint arXiv:2109.03506*.

Nickolay V. Shmyrev and other contributors. 2022. Vosk Speech Recognition Toolkit: Offline speech recognition API for An- droid, iOS, Raspberry Pi and servers with Python, Java, C and Node. https://github.com/alphacep/vosk-api.

Arjun Srinivasan, Bongshin Lee, Nathalie Henry Riche, Steven M Drucker, and Ken Hinckley. 2020a. Inchorus: Designing consistent multimodal interactions for data visualization on tablet devices. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pages 1–13.

Arjun Srinivasan, Bongshin Lee, and John Stasko. 2020b. Interweaving multimodal interaction with flexible unit visualizations for data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 27(8):3519–3533.

Arjun Srinivasan, Nikhila Nyapathy, Bongshin Lee, Steven M Drucker, and John Stasko. 2021. Collecting and characterizing natural language utterances for specifying data visualizations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–10.

Arjun Srinivasan and John Stasko. 2017. Orko: Facilitating multimodal interaction for visual exploration and analysis of networks. *IEEE transactions on visualization and computer graphics*, 24(1):511–521.

Yiwen Sun, Jason Leigh, Andrew Johnson, and Sangyoon Lee. 2010. Articulate: A semi-automated model for translating natural language queries into meaningful visualizations. In *Smart Graphics: 10th International Symposium on Smart Graphics, Banff, Canada, June 24-26, 2010 Proceedings 10*, pages 184–195. Springer.

tiangolo. 2023. tiangolo/fastapi.

Henrik Voigt, Özge Alaçam, Monique Meuschke, Kai Lawonn, and Sina Zarrieß. 2022. The why and the how: A survey on natural language interaction in visualization. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 348–374.

Henrik Voigt, Monique Meuschke, Kai Lawonn, and Sina Zarrieß. 2021. Challenges in designing natural language interfaces for complex visual models. In *Proceedings of the First Workshop on Bridging Human–Computer Interaction and Natural Language Processing*, pages 66–73.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022. Super-naturalinstructions:generalization via declarative instructions on 1600+ tasks. In *EMNLP*.

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2015. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics*, 22(1):649–658.

Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Towards a general-purpose query language for visualization recommendation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 1–6.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang,

et al. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*.

Bowen Yu and Cláudio T Silva. 2019. Flowsense: A natural language interface for visual data exploration within a dataflow system. *IEEE transactions on visualization and computer graphics*, 26(1):1–11.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Sujia Zhu, Guodao Sun, Qi Jiang, Meng Zha, and Ronghua Liang. 2020. A survey on automatic infographics and visualization recommendations. *Vis. Informatics*, 4:24–40.

A Prompt Design

The prompt is assembled as a sequence of the *visualization state* and the *table state*. Below that we put the *dialog history*, followed by a new line signaling the new input *query*. After the input query, relevant examples from long-term memory are displayed. A visual summary of the prompt design can be seen in Figure 4.

```
Visualization State:
"mark": "bar", "encoding_x_field": "city", "encoding_x_type": "nominal",
"encoding_y_aggregate": "mean", "encoding_y_field": "temperature",
"encoding_y_type": "quantitative"

Table State:
table_name : northern_european_cities
col : date (object) | temperature (float64) | radiation (float64) ...
row_0 : 2003-02-16 | -3.5 | 4.34 ...

=========

Conversation History:
INPUT: Hello
OUTPUT: action: text_response; args: "text": "Hi, how can I help?"
INPUT: show me a bar chart of the mean temperature per city
OUTPUT: action: create_vegalite; args: "mark": "bar", "encoding_x_field": "city", ...

NEW INPUT: change the map type to hybrid please

Here are some examples:
INPUT: change the leaflet map type to street
OUTPUT: action: change_map; args: "type": "street"
##
INPUT: change map type to satellite
OUTPUT: action: change_map; args: "type": "satellite"
##
INPUT: turn the leaflet map theme to dark
OUTPUT: action: change_map; args: "type": "dark"

OUTPUT: action: change_map; args: "type": "hybrid"
```

Figure 4: **Example prompt of the VIST5 language model.** *Blue:* The visualization state contains the stringified Vega-Lite specification. *Black:* The table state contains a stringified version of the column header and the first three rows of the Pandas data frame of the currently used dataset. *Green:* The conversation history contains up to eight previous turns in the dialog. *Red:* The new input field contains the current user query. *Purple:* The examples section contains up to three possible retrieved few-shot examples from long-term memory. *Orange:* The word `OUTPUT` is the last word entered into the model, signaling the start of the generation process. The subsequent action and arguments are possible outputs to be generated by the model given the preceding prompt.

B Evaluation

We evaluated the system by conducting an active user study engaging 24 users with the VIST5 dialog assistant. The user study was conducted with people of academic background (58.3% male, 37.5% female, 4.2% prefer not to say). 8.4% of the participants are in NLP, 54.2% are in Visualization, 20.8% are in climate science, and 16.6% are people from other fields subsumed under 'Others'. 62.5% of the participants were between the ages of 20 and 30, 29.2% were between 30 and 40, and 8.3% were between 40 and 50. 29.2% had less than three years of experience in their domain, 37.5% between three and five years, and 33.3% more than five years.

B.1 Method

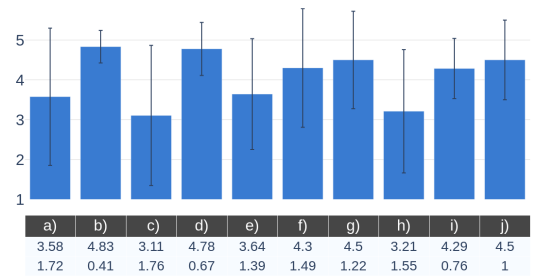
The main goal of our study was to find out:

1. The quality of the answers given by the system with respect to the different types of queries in the **NL2VIS task**.

2. The system's response quality on **few-shot tasks**.

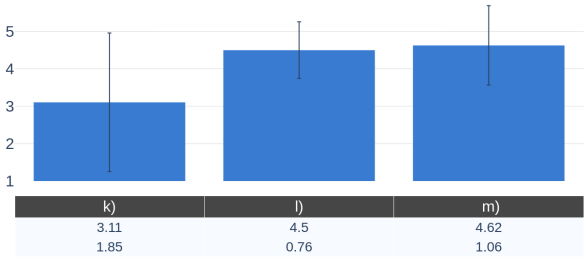
We put participants into a task-oriented dialog situation. Users were given the option to choose from a set of *seven* different climate data sets. To generate goals for users to achieve with the system, we generate visualization tasks from the pool of common low-level visualization tasks specified by Amar et al. (2005): : *characterize distribution, compute derived value, correlate, determine range, filter, find extremum, find anomalies, cluster, retrieve value, sort*. Every user is randomly assigned **two** of those tasks. A low-level visualization task is presented to the user as a general instruction, e.g., to filter the dataset according to a certain condition. The user must then try to solve the task by interacting with the chatbot. Further, every participant was assigned **one** few-shot task from the pool of few-shot categories: *custom visualization, custom functionality, locomotion* which each is comprised of several few-shot tasks, but we are mainly interested in the response quality per category. The custom visualizations that can be created are *marker plots*, *heat maps*, *flow visualizations*. Custom functions to be invoked include *exporting* visualizations, *changing map style*, and *updating* the dataset. Locomotion few shot tasks include *zooming in/out*, *moving left/right/up/down*, and *navigating* to a city of choice. To solve a task, a user can ask as many questions as necessary. During the interaction, users are prompted to rate the quality of each response from the chatbot on a Likert scale from 1 (poor) to 5 (very good), i.e. how appropriate the response was given the query. In addition, users are asked to provide textual feedback on what they consider to be particularly good or bad answers. This helps us understand these extreme cases better in hindsight and learn from them. Before the study began, users were shown a video of a short sample conversation (less than 10 turns) between a user and the chatbot, explaining how to rate responses and where to provide feedback.

Once all tasks have been completed, we allow the participants to explore the system freely in an



a)	b)	c)	d)	e)	f)	g)	h)	i)	j)
3.58	4.83	3.11	4.78	3.64	4.3	4.5	3.21	4.29	4.5
1.72	0.41	1.76	0.67	1.39	1.49	1.22	1.55	0.76	1

Figure 5: Results of the user evaluation on the ten low-level tasks of Amar et al. (2005): *a) characterize the distribution, b) compute a derived value, c) correlate, d) determine range, e) filter, f) find extremum, g) find anomalies, h) cluster, i) retrieve value, j) sort*. The *mean* is provided in the first row of the table below, *std* in the second.



k)	l)	m)
3.11	4.5	4.62
1.85	0.76	1.06

Figure 6: Results of the user evaluation on the three few-shot task categories: *k) custom functionality, l) custom visualization, m) locomotion*. The *mean* is provided in the first row of the table below, *std* in the second.

unbounded way. The unconstrained interaction helps us get additional feedback for a broader horizon of uses that we may not have thought of before. This feedback is interesting for guiding future work.

B.2 Results

All in all, we collected a set of 279 dialog turns from the users during the study. The average dialog has a number of 11.6 user turns, which is higher than the average number of user turns in current task-oriented dialog datasets such as MultiWOZ (Budzianowski et al., 2018).

NL2VIS Tasks. The results on the low-level visualization tasks are shown in Figure 5. The mean Likert score across all tasks is 3.82. The standard deviation across all tasks is 1.53. The mean for each task is shown in the first row of the table in Figure 5, and the standard deviation is shown in the second row. We can see that the mean score for the tasks *compute derived value, determine range, find extremum, find anomalies, retrieve value* and *sort* is very high, with an average value above 4. This tells us that the system provides high-quality responses for these subsets of low-level visualization tasks.

Tasks like *characterize distribution, correlate, filter* and *cluster* have an average value above 3, but also show a larger standard deviation. This shows that for these tasks the response quality varies more between appropriate and inappropriate responses, but the tendency is towards positive responses. Overall, the system does not perform below average on any of the tasks.

Few-Shot Tasks. The results on the few-shot tasks are shown in Figure 6. The average rating over all tasks is 3.77. The standard deviation over all tasks is 1.65. The mean for each task is shown in the first row of the table in Figure 6, and the standard deviation is shown in the second row. We can see that the means for the *custom visualization* task and the *locomotion* task are very high with values above 4. This shows that the system had no problems finding out how to create custom visualizations on the leaflet map and navigating it based on a few examples. The mean scores for the *custom functionality* task are above 3 and show higher standard deviations, indicating that the response quality is more variable for this few-shot category. We found a possible explanation for this in the vulnerability of the few-shot paradigm to typos. In particular, typos when changing the map type or selecting column names cause problems because the system usually passes the arguments as they are given in the input to the function, which then leads to errors in execution. The integration of a spell checker or the use of system-initiated check questions in case of uncertainty are possible levers for future improvements in this respect.

Overall, the system always scores above the mean of 3 for all tasks. This shows that, on average, users found the responses to be helpful. However, it also shows that while the system performed well on the majority of responses, it did not perform optimally on all inputs.