Benchmarking Natural Language to Data Visualization (NL2Viz) Models

Enrique Casillas
MIT
enriquec@mit.edu

ABSTRACT

Data visualization is an incredibly important tool used to convey information to people. One new method of creating data visualizations is through *NL2Viz* models, which take in natural language queries as inputs and return some visualization object, usually in the form of a Vega-Lite specification. Two *NL2Viz* models, ncNet and nl4dv, were analyzed and evaluated using the SSIM metric to directly compare produced visualization images to a benchmark dataset. After evaluation it was found that ncNet outperforms nl4dv in terms of producing accurate high-quality visualizations based on natural language queries. This project also presents an online tool, *nl2viz*, which was created to view these comparisons manually. The code used for both the evaluation and the online tool can be found at https://github.com/casillasenrique/nl2viz.

1 INTRODUCTION

Data visualization is an incredibly important tool used to convey information to people. It can be used to drive important decisions, win arguments, or simply showcase something interesting for people to see. Much work has therefore been put into facilitating the creation of visualizations, from Python libraries like Seaborn [6] to low-code, user-friendly platforms and applications such as Tableau [5]. One emerging method of data visualization is using natural language processing (NLP) to automatically create a visualization based on a single human-readable query (referred to as *NL2Viz* in this paper). Users could ask a system a question such as "show me the top 10 cinemas by location," and it could understand the query and produce a data visualization automatically with zero code.

As these technologies emerge, they will need to be evaluated against some ground truth of data visualizations. Today most new systems are evaluated either by seeking experts to manually rank produced visualizations, or no evaluation is included at all [2]. This project seeks to create an easy way to compare a model's output with some benchmark visualization, as well as provide some alternatives for evaluation. As such the project comes in two parts. First, the project performs a comparison between two different *NL2Viz* models against a common benchmark dataset using certain metrics. Second, the project provides a proof-of-concept web-based tool, *nl2viz* , that allows users to manually generate these visualizations and compare them against the benchmark in an intuitive manner, without needing code.

2 RELATED WORK

This section discusses related projects and research in the field of *NL2Viz*.

2.1 Evaluation

nvBench is existing research that recognizes the difficulty in evaluating these kinds of *NL2Viz* models. The paper, written by the Tsinghua Database Group (TDG), presents a benchmark dataset of visualizations that can be used to test *NL2Viz* models and evaluate how "good" they are. The benchmark, used in this project, provides a large set of $(nlQuery, viz)$ pairs, where $nlQuery$ is simply a human-readable natural language (NL) query asking for some sort of visualization, and viz is the supposed "gold-standard" visualization that should be produced. These visualizations were created based on an existing model of converting NL queries to SQL queries, then visualizations were produced and checked over by 23 field experts with visualization experience [2].

2.2 Online Tool

nl4dv, created by Georgia Tech researchers, provides an existing open-source *NL2Viz* online platform called *Vega-Lite Editor* [4]. This platform can only use the single nl4dv model, and its purpose is mainly to create visualizations based on natural language queries. This project's online platform takes a different approach in also providing a comparison against a benchmark visualization.

3 NL2VIZ MODELS

NL2Viz models can be defined as functions that take in a CSV dataset and a string natural language query, and output a Vega-Lite specification. Models can also have state, which provides many additional features not explored in this paper, such as dialogue-based visualization generation [4]. However, one important state that this project takes advantage of is the idea of a working dataset. An *Nl2Viz* model provides a function, `change_dataset()`, that allows users to switch what dataset is being used to generate visualization, that is, switch the application context.

This project focuses on two *NL2Viz* models. The first is NL4DV, a toolkit provided by Georgia Institute of Technology researchers, which uses attribute inference and explicit task inference to convert a parsed query into a Vega-Lite specifications [4]. The second is NCNET, TDG's own *NL2Viz* implementation, which uses a Transformer-based model (translation) to generate Vega-Lite specifications from a query [3]. Both NL4DV and NCNET implement this high-level definition of an *NL2Viz* model outlined above (for instance, both of them are stateful and allow for this switching of datasets). The most important point however is that both produce a Vega-Lite specification as output to a natural language string query. This is important for consistency when comparing them against a single benchmark.

4 EVALUATION METHODOLOGY

The section covers the methodology used in the evaluation portion of this project. Recall that the goal of this evaluation was to get some sense as to which of the two models, NCNET or NL4DV, performs "better" overall by seeing which model produces the "better" visualizations given natural language queries. A large portion of the evaluation was done with the help of NVBENCH, a benchmark containing a large dataset and $(nlQuery, viz)$ pairs [2].

4.1 Cleaning the Datasets

The datasets used in this evaluation were taken from the NVBENCH dataset (found here), which itself is adapted from the Spider benchmark that contains 200 databases across 138 categorical domains (e.g. cinema, aircraft, etc.) [7]. One problem with the provided dataset was that they are all provided in `.sqlite` formats, with the possibility of having multiple tables. Although NCNET supports SQLite datasets, NL4DV does not—it only supports single-table CSV style datasets. As a result, each database had to be manually converted into a CSV, which simply involved iterating through all tables in each database and converting each table to a dictionary which could be saved as a CSV. The following Python code shows a snippet of this process:

```
# Fetch all tables from the .sqlite database
table_names = cur.execute(
    "SELECT name FROM sqlite_master WHERE "
    "type='table' ORDER BY name;"
).fetchall()

# Extract all of the content for each table
data = {}
for (table,) in table_names:
    cursor = cur.execute(f"SELECT * FROM {table}")
    column_names = [
        description[0]
        for description in cursor.description
    ]
    data[table] = [
        dict(zip(column_names, row))
        for row in cursor.fetchall()
    ]
conn.close()

# Write the data to a CSV file for each table
...
```

Unfortunately, many NVBENCH benchmark queries are based on multiple tables within a single dataset; that is, the benchmark expects *NL2Viz* models to be able to perform table joins. By the nature of its single CSV expectation, NL4DV is unable to do these types of joins. Therefore, some of the resulting CSVs that were generated could not be used, since there were no benchmarks associated with them on their own. In the end, 141 CSV files were generated.

4.2 Cleaning the Benchmarks

Next, the crucial step of matching benchmark NL queries with their corresponding visualizations was performed. As explained in Section 3, the visualizations in this project expect the use of Vega-Lite visualization grammar. Unfortunately, although NVBENCH provides a JSON object mapping NL queries to generic visualization objects, it does not provide a mapping between NL queries and Vega-Lite specifications. This is provided implicitly in 7,247 HTML files that generate a webpage view of the benchmark and corresponding dataset [2]. As such, a manual process of scraping each HTML page to collect the $(nlQuery, vegaSpec)$ pairs was performed in Python with the help of the BeautifulSoup library (a small snippet of this is shown below). Because of the large size of each benchmark, this process was done in batches and then merged into a single file. Note that the merging process removed all benchmarks that required the use of more than one table/CSV, as explained in Section 4.1.

```
with open(filepath, "r", encoding="utf-8") as f:
    contents = f.read()
    soup = BeautifulSoup(contents, "html.parser")

    # Find the SQL query and get the tables used
    p_with_sql_query = [
        p
        for p in soup.find_all("p")
        if p.text.startswith("Visualize")
    ][0]
    sql_query = p_with_sql_query.text
        .replace("Visualize", "").strip()
```

```
# The last <script> tag is the one that
# contains the Vega-Lite spec
script_contents = str(soup.find_all("script")[-1])

# Extract the spec by finding the opening
# and closing brackets
start_index = script_contents.find("vlSpec1  =")
end_index = script_contents.rfind("}")
vega_lite_spec = (
    script_contents[
        start_index : end_index + 1
    ]
    .replace("vlSpec1  =", "")
    .strip()
)
vega_lite_spec = ast.literal_eval(vega_lite_spec)
```

The result of this process was the creation of a ~10MB `benchmark_meta.json` file, which has as a dictionary structure mapping a benchmark ID to a benchmark object defined as follows:

```
{
    "tables_used": list[str] # Tabled used
    "nl_queries" list[str] # List of queries
    "vega_spec": dict # The Vega-Lite spec
}
```

To help with the evaluation process (explained below), a `dataset_to_queries.json` lookup table was created using the benchmark metadata in order to easily find all of the NL queries associated with any given dataset.

Lastly, nvBench provides a rating of "hardness" or difficulty for each NL query. This rating gives a rough sense as to how difficult the query is to understand and successfully convert into a visualization. The queries are grouped into 4 ratings, *easy*, *medium*, *hard*, and *extra-hard*.

4.3 Evaluation Process

Once the datasets and benchmarks had been set up, the actual evaluation process could begin. An image similarity measure was used for the actual comparison between a model visualization result and the benchmark visualization. Additionally, because of the relatively large number of datasets and queries and the computational requirements in running these models, the Dask library was used to speedup the computation using parallel computing.

4.3.1 SSIM. This paper proposes the use of existing image quality and similarity metrics to compare visualizations against each other. In this application, the structural similarity index measure (SSIM) is used to try to do just that. SSIM is a model that takes two images and determines their similarity by calculating the differences in "structural information" [1]. Structural information, in this context, provides the notion that nearby pixels have some form of interdependence, which can be well suited for comparing highly structured images such as visualizations. Although this metric is primarily used to measure image quality, such as in image steganography, it also provides a decent image similarity interpretation, where a value $SSIM(img1, img2) = 1$ means that the images are identical. Therefore, for each pair of model-produced visualization $modelViz$ and benchmark visualization $benchViz$, $SSIM(modelViz, benchViz)$ provides a decent measure of model performance, and thus was used in this project's evaluation. Note that the scikit-image library was used to implement this.

4.3.2 Evaluation Pipeline. This section discusses the steps taken to evaluate the models on the benchmark. The evaluation process itself posed several challenges at different steps that needed to be addressed.

(1) As discussed in Section 4.2, a lookup table `dataset_to_queries.json` was created by iterating through all of the benchmarks and storing a mapping between dataset names and a list of NL queries that require that dataset. This information was saved in memory in a local variable `dataset_to_queries_lookup`.
(2) Using the lookup's keys, a list of `(model, dataset_name)` pairs was created for a total of 2 models · 141 datasets = 282 pairs, stored in the `parameters` variable. The Python code for this is shown below.

```
parameters = [
    (model_name, dataset_name)
    for model_name in ["ncNet", "nl4dv"]
    for dataset_name in dataset_to_queries_lookup\
        .keys()
]
```

(3) Using the Dask library, the computation on the `parameters` list was distributed across 12 workers and computed. The actual code that executes this computation is outlined below. Note that the `compute_metrics` function is the main function that performs the evaluation on each of these pairs.

```
lazy_results = []
for i, (model_name, dataset_name) in enumerate(
    parameters
):
    lazy_result = dask.delayed(
        compute_metrics
    )(model_name, dataset_name)
    lazy_results.append(lazy_result)
```

```
result = dask.compute(*lazy_results)
```

(4) The `compute_metrics` function is then performed which calculates the metrics on all `(model_name, dataset_name)` pairs. The contents of this function are described below.
  (a) For each of the `(model_name, dataset_name)` pairs, the list of queries to ask was generated by fetching `dataset_to_queries_lookup[dataset_name]`.
  (b) Next, the benchmarks were retrieved using the `benchmark_meta.json` file; the filtered list of benchmarks was saved to the variable `benchmarks_with_dataset`.
  (c) Next, the model instance itself was created, either an NCNET or NL4DV instance.
  (d) Then, for each NL query, the following procedure was followed:
    (i) Given a (*model*, *dataset*, *nlQuery*) triplet, the model's Vega-Lite specification was created by asking it to produce a visualization given *dataset* and *nlQuery*. The benchmark's visualization was retrieved by simply searching for the corresponding NL query within the list of current benchmarks for the dataset, `benchmarks_with_dataset`.
    (ii) Given the resulting (*modelSpec*, *benchmarkSpec*), the actual visualization images were generated and compared. This was a major hurdle that had to be addressed, since there is currently no native Python support for converting Vega-Lite specs to image files (which is required for the SSIM metric). Fortunately, there is a workaround where both specs can be saved to JSON files, then those files are read and converted and saved to `.png` files using the vega-lite node module's `vl2png` CLI tool. Lastly, the `.png` files are read using the Pillow image processing library, converted to $500 \times 500 \times 3$ `numpy` arrays (with the 3$^{\text{rd}}$ dimension being the RGB channels), and compared using `skimage.metrics.structural_similarity()`. Using this method, the SSIM metric was returned.
  (e) Following each query, datapoints were saved as dictionaries containing the query, whether or not the model was able to produce a visualization, and the SSIM value.
(5) All of the data points for each query were gathered into a single dictionary and saved in JSON format.

Once completed, a set of 276 JSON files were saved containing evaluation metrics for each (*model*, *dataset*) pair (note that 6 files were not produced due to trouble loading in the datasets, see Section 5). A final CSV was produced called `clean_evaluations.csv`. The CSV columns are described in Table 1.

Model	Dataset	NL Query	Hardness	Produced Viz	SSIM
The model name	The dataset name	The asked natural language query	Difficulty measure	1/0 produced a visualization	The SSIM value compared to benchmark
ncNet	accounts	"Show..."	Easy	1	0.743

Table 1: The evaluation primary CSV file (with example row)

The results of this evaluation are explained in the next section.

5 RESULTS

Figure 1 shows an overview of the results. A total of 18,490 queries were analyzed across the two models, each grouped into their difficulty category. As the chart shows, NCNET yielded slightly more results due its ability to produce more visualizations. The `basketball_match` dataset had the largest number of queries.

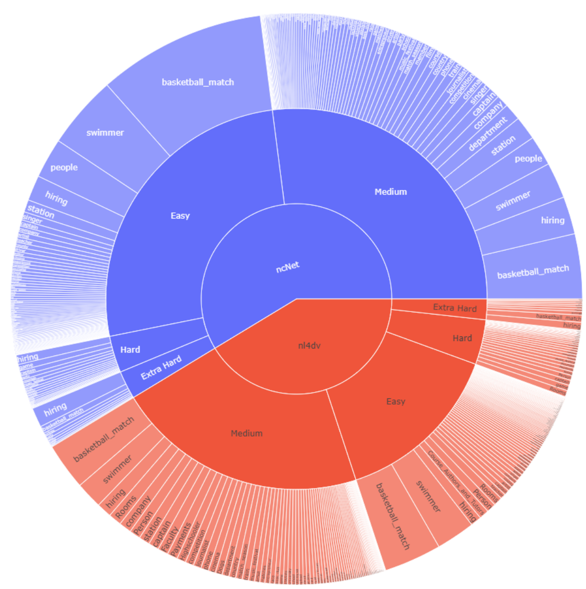

Figure 1: A summary of the results showing the distribution of difficulty levels and datasets across both models.

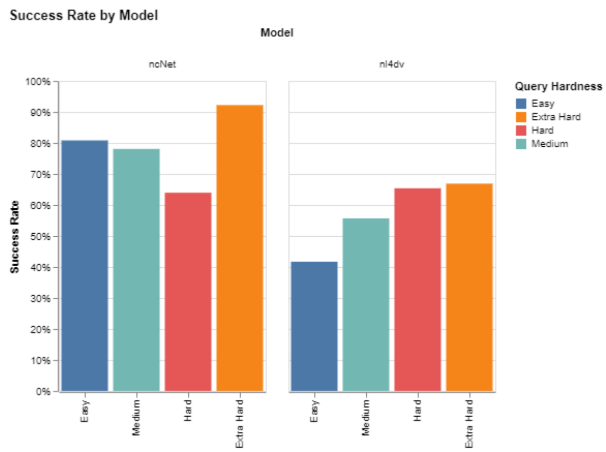

Figure 2: The success rates of each model grouped by query difficulty/hardness.

5.1 Success Measure

The first metric that was used was a simple "success" measure. Both models ran into scenarios where they could not understand an input query, and failed to produce a visualization. Therefore, a simple test to see which model performed better was to simply find the percentage of queries that the model could understand and produce a valid visualization object. Figure 2 shows the results of this analysis. ncNet had a higher success rate on three difficulty categories than nl4dv, meaning it was able to understand more queries. nl4dv could understand hard-difficulty queries slightly better. One observed peculiarity was the increase in nl4dv's success rate as queries got more difficult. One possible explanation for this is that many of the easy and medium queries asked for pie charts, which nl4dv did not seem to be able to understand.

5.2 SSIM Comparison

Taking all of the results that *did* produce visualizations, the second metric was the actual SSIM value, which involved comparing the model-produced visualization with the benchmark visualization. Figure 7 (at the end of the paper) provides a summary of the average SSIM values for both models across all of the datasets. Note that the datasets are only shown if both models were able to produce visualizations with them. Here we see that for many datasets, ncNet achieved higher average SSIM values than nl4dv, though there were a few datasets such as `city` in which nl4dv performed better overall.

Figure 3 shows an aggregated summary of the average SSIM values in a similar manner as the success rates. It shows the average SSIM yielded by visualizations grouped by the query difficulty level for both models. As with the success rates, ncNet performed better overall in all categories except the hard queries. Again, there is an upward trend in average SSIM values for nl4dv, which can similarly be attributed to the lack of model training for super simple pie or bar charts.

5.3 Qualitative Analysis

Here are some findings and comments made while working with the two *NL2Viz* models.

ncNet had a lot of trouble working with many CSV datasets. In particular, if a dataset had the same name but different capitalization, ncNet failed to recognize the difference, due to it saving mappings to a file while converting all names to lowercase. In this way it also fails to be modular, since it has to consult many central databases and lookup tables which are mutated as more queries are made. ncNet, unlike nl4dv, also does not come with a PyPi package implementation, and has to be copied over directly from source which can be a major pain point.

nl4dv overall performed worse using the quantitative metrics discussed above. One major reason for this seems to have been that the model does not understand very simple queries such as those requiring pie charts. nl4dv is also less advanced in that it only knows how to work with a single CSV dataset at a time, and cannot perform table joins in order to create more complex visualizations.

6 ONLINE TOOL SYSTEM DESIGN

Because the evaluation itself does not do a great job of showing the actual produced visualizations, this project also consists of an online platform that allows users to view them side by side with the corresponding benchmark. The general workflow of this website is as follows:

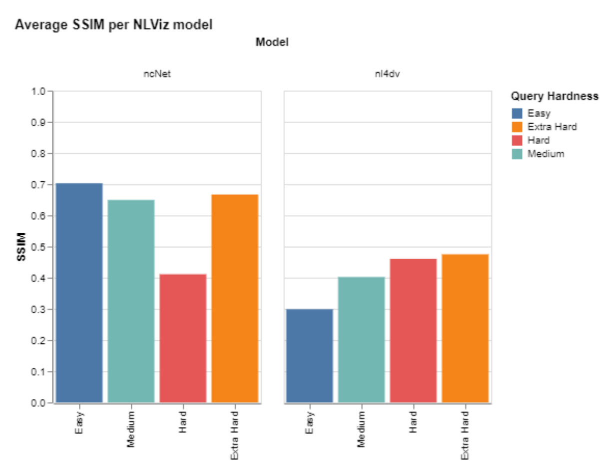

Figure 3: The average SSIM values for each model grouped by query difficulty/hardness.

(1) Users select an NL2Viz model to use (one of NCnet, nl4dv).
(2) Users select a dataset to use, one of the CSVs provided by the benchmark.
(3) Users select a query from the benchmark, or enter their own custom query.
(4) The model's visualization is shown next to the corresponding benchmark, if it exists. Users can view the visualization in a Vega editor, and have access to the spec along with some extra information, like the visualization attributes.

As far as the overall system design, the online tool is split into a standard server/client architecture, explained in the following sections. Figure 4 shows a diagram of the entire system.

6.1 Backend

Because of the dominance of Python in the data analysis and natural language processing field, the server is written in Python 3.9 using the Flask framework. The server is tasked with maintaining a list of connected clients, where each client stores a working *NL2Viz* model and a dataset in state. The server then provides several API endpoints that allow the client to communicate with these models and produce results. For example, users can switch datasets using the `POST /api/dataset` route, and execute queries against the benchmark with the `GET /api/benchmark/execute?query=<query>` route. Table 2 lists all of the API endpoints with brief descriptions.

As per the system diagram, the datasets are provided from nvBench and are the same datasets used in the evaluation

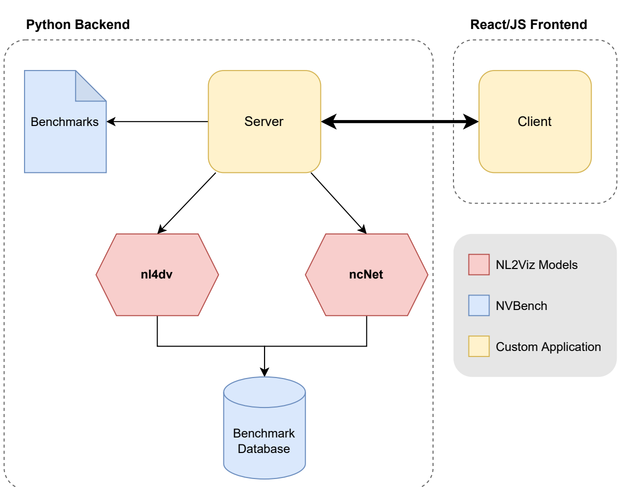

Figure 4: *nl2viz* online tool system diagram.

API Endpoint	Description
GET datasets	Returns all available datasets
GET datasets/<dataset>	Get a particular dataset by name
GET dataset	Gets the user's current dataset
POST dataset	Switch the current dataset
GET models	Returns all available *NL2Viz* models
GET model	Gets the user's current model
POST model	Switch to a different *NL2Viz* model
GET benchmark /<dataset>/queries	Get the benchmark's NL queries for the given dataset.
GET benchmark/execute	Executes an NL query and returns the result and the benchmark visualization
GET execute	Only executes the query, does not try to find the benchmark
GET client	Gets the current client session
GET clients	Returns a list of all connected clients

Table 2: *nl2viz* API Endpoints

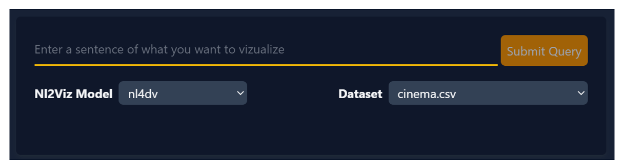

Figure 5: *nl2viz* query input box.

(these are the 141 CSV files). The benchmarks are stored in the `benchmark_meta.json` file as explained in the evaluation section (Section 4) and are fetched on a per-request basis.

6.2 Frontend

The frontend consists of a single client application created with React, and serves as a way for a user to interact with the server. As hinted at in the workflow, the client contains selection dropdowns to select an *NL2Viz* model as well as a dataset pulled directly from the server. It also provides an input box to enter queries. All of these selections are grouped into a query input box as shown in Figure 5.

The bottom half of the UI (the dashboard) contains 3 panels, shown in Figure 6. The leftmost panel shows the raw model output based on an NL query, which most importantly contains the raw Vega-Lite specification. The rightmost panel contains the benchmark output and specification, if it exists. It also contains some additional visualization information such as attributes shown. The center panel is a split view between the model's visualization output on the left and the benchmark visualization on the right. Related queries are shown above this split view, and underneath the model's

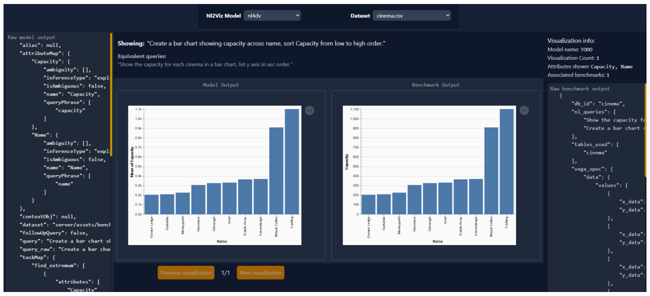

Figure 6: The *nl2viz* dashboard.

output there are buttons to toggle between the model's visualizations in case it produced more than one.

As for the client structure, the bottom half of the UI is contained in the React component `Dashboard.js`. The query form shown is contained in separate React component `QueryForm.js`. The actual home page is contained in `Home.js` and stores the client state.

7 FUTURE WORK

This project opens up some possibilities for future work. On the evaluation side, better metrics can be used to compare the model visualizations with the benchmark. It would be interesting to see new metrics being developed for this emerging field, either as completely new ideas or some combination of existing metrics. Of course, additional *NL2Viz* models could be found and put through this evaluation pipeline.

As for the online tool, it would be helpful to allow users to view a snapshot of each dataset they select in order to get a better idea of what queries they can ask. Another feature that could be added would be on-demand SSIM metrics; essentially compute the measure on demand and display it next to the benchmark visualization if it exists, to further allow users to evaluate these models. Lastly, it could be helpful to integrate a specification editor in case the model almost got the visualization right, but may need a few tweaks in order to perfectly match the benchmark.

8 CONCLUSION

This project presented an evaluation of two natural language to visualization models, NCNET and NL4DV. Using a success score and SSIM metrics, it found that NCNET performed better overall in producing accurate visualizations when compared to a benchmark dataset. A new online tool, *nl2viz*, was presented which allows users to manually provide queries to either of these models and immediately see the differences between the model's output visualization and an associated benchmark.

9 ACKNOWLEDGEMENTS

Special thanks to professor Sam Madden for feedback on this project. Additionally, I would like to thank professors Madden and Kraska, as well as Markos Markakis and Amadou Ngom for an instructive and engaging semester.

REFERENCES

[1] Dominique Brunet, Edward R. Vrscay, and Zhou Wang. On the mathematical properties of the structural similarity index. *IEEE Transactions on Image Processing*, 21(4):1488–1499, 2012.

[2] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. Synthesizing natural language to visualization (nl2vis) benchmarks from nl2sql benchmarks. 2021.

[3] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2021.

[4] Arpit Narechania, Arjun Srinivasan, and John Stasko. NL4DV: A Toolkit for generating Analytic Specifications for Data Visualization from Natural Language queries. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2020.

[5] Salesforce. Tableau, 2022.

[6] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[7] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *CoRR*, abs/1809.08887, 2018.

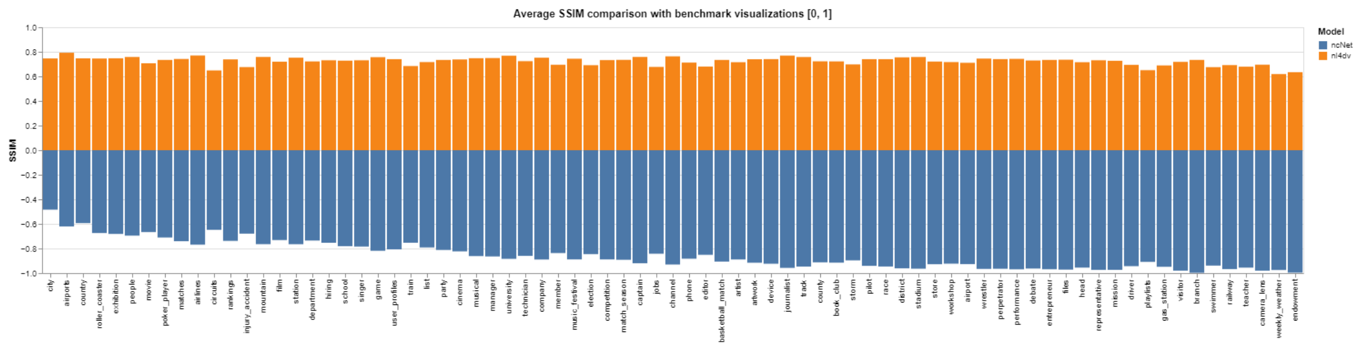

Figure 7: Overview of SSIM results, average SSIM by model and dataset.