Kanit Wongsuphasawat (Research Statement)

Visualization is a critical tool for data science. Analysts use plots to explore and understand distributions and relationships in their data. Machine learning developers also use diagrams to understand and communicate complex model structures. Yet visualization authoring requires a lot of manual efforts and non-trivial decisions, demanding that the authors have a lot of expertise, discipline, and time in order to effectively visualize and analyze the data.

My research in human-computer interaction focuses on the design of tools that **augment visualization authoring with automated design and recommendation**. By *automating repetitive parts* of authoring while *preserving user control* to guide the automation, people can leverage their domain knowledge and creativity to achieve their goals more effectively with less efforts and human errors. In my thesis, I have developed new formal languages and systems for chart specification and recommendation, and used them to develop graphical interfaces that enable new forms of recommendation-powered visual data exploration [1-4]. I also built a tool that combines automatic layout techniques with user interaction to help developers visualize and inspect the structure of deep learning models in TensorFlow [5]. These systems have been open sourced and adopted in data science communities.

Augmenting Exploratory Data Analysis with Visualization Recommendation

Exploratory analysis of previously unseen data involves both open exploration and question answering. While tools like Tableau and ggplot2 support a variety of charts for question answering, they typically require *manual chart authoring*, involving many decisions including choosing data fields, applying data transformations, and designing visual encodings. This manual process can be tedious and non-trivial, demanding familiarity with the data domain as well as analysis and design expertise. While analysts should achieve systematic data coverage in their exploration, in practice they may overlook important insights, such as potentially confounding factors and data quality issues, or prematurely fixate on specific questions due to the lack of expertise or discipline.

My thesis explores how to design interactive systems that **complement manual chart authoring with chart recommendation to facilitate rapid and systematic exploration** of tabular data. I have developed new languages and systems for chart specification and recommendation [1-4], and used them to build graphical interfaces that enable new forms of recommendation-powered visual data exploration [1,2] (Figure 1). The **Vega-Lite** visualization grammar [1,3] provides a representation for specifying and reasoning about charts. The **CompassQL** query language and recommender engine [2,4] provide a generalizable framework for chart recommendation via queries over the space of visualizations. With Vega-Lite and CompassQL as the foundations for chart recommendation, I used the iterative design process to develop and study new recommendation-powered graphical interfaces. **Voyager** [1] enables data exploration via browsing of recommended charts. Our user study, which compared Voyager with a chart authoring tool, indicated the complementary benefits of manual authoring and recommendation browsing. Inspired by the study result, **Voyager 2** [2] blends manual and automated chart authoring to facilitate both question answering and exploration in a single tool. Besides Voyager, Vega-Lite and CompassQL have also enabled other applications and research projects.

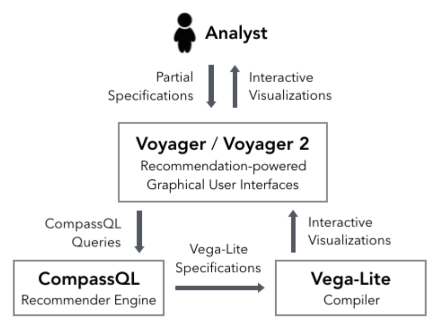

Figure 1: The Vega-Lite grammar and CompassQL query language enable new recommendation-powered visualization tools including Voyager and Voyager 2.

Vega-Lite: a Grammar of Interactive Visualizations

Building a chart recommendation engine requires a representation for enumerating and evaluating candidate charts. To enable a broad range of recommendation, the representation must be *expressive*, supporting a variety of charts. To facilitate reasoning in the recommender engine, the representation must also be *concise*, requiring a small number of properties to be determined by the engine.

To provide a representation that satisfies these criteria, we developed Vega-Lite [1,3], a high-level visualization grammar built on top of Vega [6]. As a grammar, Vega-Lite provides primitive building blocks for composing an *expressive* range of charts. A *single-view specification* in Vega-Lite (Figure 2) describes `data` sources, `mark`, and `encoding` mappings from (optionally transformed) data `fields` to visual channels such as `x`, `y`, or `color`. Inspired by existing grammars for visual analysis like ggplot2 and Tableau's VizQL, Vega-Lite provides a *concise* syntax by automatically generating low-level chart components such as scales and guides, and allows users to customize these components by overriding their default properties. With a concise JSON syntax, Vega-Lite enables both rapid manual authoring and programmatic recommendation of charts.

```
{
  "data": {"url": "flights.json"},
  "mark": "bar",
  "encoding": {
    "x": {
      "bin": true,
      "field": "distance",
      "type": "quantitative"
    },
    "y": {
      "aggregate": "count",
      "type": "quantitative"
    }
  }
}
```

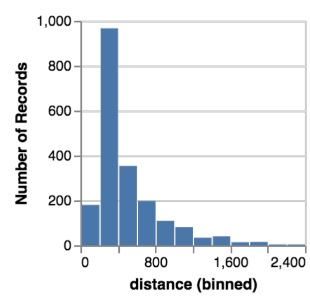

Figure 2: A histogram in Vega-Lite: bars that map a binned field and aggregated count to their position and length.

Beyond existing grammars, my colleagues and I have extended Vega-Lite to support *interactive, multi-view graphics* [3]. A novel *view algebra* enables hierarchical composition of layered and multi-view plots via operators including *facet*, *layer*, *concatenation*, and *repeat*. Interactions can be defined by specifying and applying *selections*, an abstraction that defines input event processing, points of interest, and a predicate function for inclusion testing. With these building blocks, Vega-Lite enables concise specification of interactive multi-view plots. For example, coordinated histograms in Figure 4 can be defined within a few dozen lines of JSON, compared to at least a few hundred lines of code in other libraries like Vega [6] and D3.

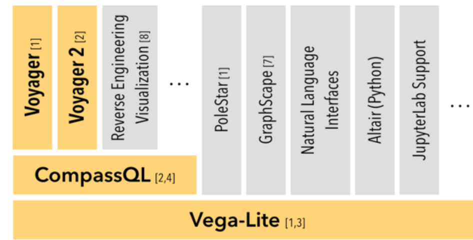

Figure 3. Vega-Lite has served as a platform for developing other applications and research projects. Highlights are my thesis contributions.

In addition to supporting chart recommendation in my thesis, Vega-Lite has served as *a platform for developing other applications and research projects* (Figure 3). We used Vega-Lite to develop an automatic model to reason about visualization similarity and sequencing [7]. Our colleagues at Stanford [11], Georgia Tech, and Princeton are using Vega-Lite to build natural language interfaces for data visualization and analysis. As a declarative format, Vega-Lite also enables *sharing across applications and platforms*. JupyterLab, the latest version of the Jupyter/iPython notebook, supports Vega-Lite and Vega as its official plotting formats. Vega-Lite is also wrapped as native visualization libraries in many languages. For example, our colleagues have built a Python wrapper called *Altair* and noted that *"Vega-lite (and Vega) are perhaps the best existing candidate for a principled lingua franca of visualization"*. With an easy-to-use yet flexible design, Vega-Lite is also used for *teaching* in a book for practitioners [13] and in classes at leading institutions including Stanford, CMU, and the Universities of Maryland and Washington. Vega-Lite has over **1,000 GitHub stars** and **30,000 downloads per month** on NPM.

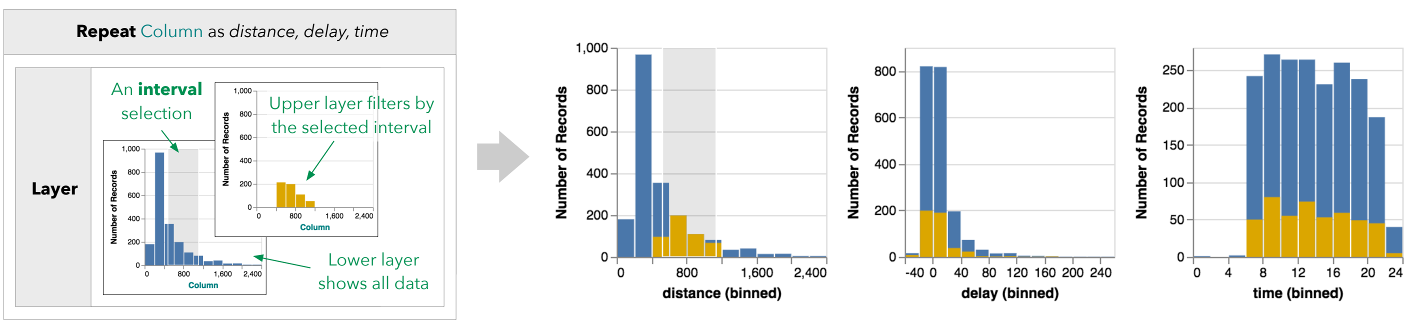

Figure 4: A column-based repetition of a layered histogram with a selection produces coordinated histograms. For each subplot, the lower layer (blue) shows the full dataset with an interval selection on x-axis while the upper layer (gold) shows the data in the selected range.

CompassQL: Visualization Query Language & Recommender Engine

Prior projects on chart recommendation typically develop customized engines to suggest chart designs for a given data or suggest potentially interesting data for a fixed chart template, without giving users fine-grained control over the suggestions. To provide a more general-purposed framework for recommendation [2,4], I developed the CompassQL query language and recommender engine, which facilitate chart recommendation via *queries over the space of visualizations*.

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "x",
      "field": "Horsepower",
      "type": "quantitative"
    },{
      "channel": "y",
      "field": "MilesPerGallon",
      "type": "quantitative"
    }]
  },
  "chooseBy": "effectiveness"
}
```

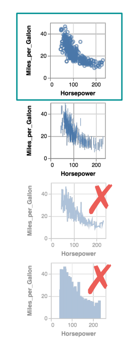

Figure 5: A CompassQL query for suggesting mark type for the shelf interface in Figure 7. As the `mark` is a wildcard, CompassQL enumerates marks, applies built-in constraints to prune misleading marks, and chooses the best mark based on a perceptual `effectiveness` ranking.

CompassQL represents a chart recommendation query in the form of a *partial chart specification*, which omits some properties to be suggested by the engine. A specification in a CompassQL query (Figure 5) has a structure like a single-view specification in Vega-Lite, but can contain *wildcards* to indicate properties that should be suggested by the engine. To organize outputs, a query may contain *recommendation directives* for grouping redundant plots (such as plots with similar data fields and transformations) and choosing or ordering plots (e.g., by an empirically-derived perceptual `effectiveness` ranking of visual encodings).

CompassQL recommends charts for a given query by reasoning about candidate chart specifications. To enumerate candidate specifications, CompassQL replaces each wildcard in the query with concrete values that satisfy both user-defined constraints in the query and the engine's built-in constraints, which applies visualization design knowledge to prune misleading charts. The engine then groups and ranks candidate charts based on the recommendation directives.

To assess if CompassQL enables more general-purposed chart recommendation, I have demonstrated that CompassQL queries can express a variety of existing recommendation approaches [4], including techniques used within Tableau [9] and recent work from the database community [12]. More importantly, I have used CompassQL as a framework to develop novel recommendation-powered interfaces for visual data exploration, including the Voyager visualization browser [1] and Voyager 2 [2], which blends chart specification and recommendation in a single tool. CompassQL was also used to generate training data for an analysis pipeline that reverse-engineers visual encodings from bitmap chart images [8].

Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations

We used Vega-Lite and CompassQL as the foundations to develop new recommendation-powered graphical interfaces for exploratory analysis. As manual chart authoring can impede rapid and systematic exploration, we developed Voyager [1], a system that facilitates data exploration with interactive navigation of recommended charts (Figure 6).

To facilitate rapid and systematic exploration, Voyager embeds analysis and visualization design practices to *guide exploration while preserving user control*. To encourage analysts to thoroughly examine the data and avoid premature fixation, Voyager shows univariate summaries of all fields upon loading a new dataset. As exploration proceeds, users can focus on specific aspects of the data and steer the recommendations by selecting data fields and transformations (Figure 6, left). To promote broad exploration, Voyager prioritizes showing *data variation* (different fields and transformations) over *design variation* (different visual encodings of the same data). Besides charts showing selected fields (Figure 6, top right), Voyager also suggests charts with one extra field to help analysts consider other potentially relevant fields (Figure 6, lower right). For each suggestion, Voyager applies the perceptual effectiveness ranking in CompassQL to pick the best visual encoding.

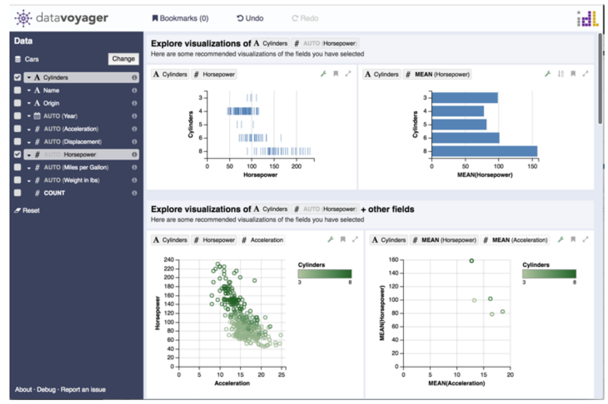

Figure 6: Voyager facilitates systematic and rapid data exploration by presenting recommended charts based on analysis and visualization design practices for users to browse (right). To focus on specific aspects of the data and steer the recommendations, users can select data fields and transformation functions (left).

We evaluated Voyager with a user study on exploratory analysis of previously unseen data. To provide a baseline system, we built *PoleStar*, a chart authoring tool modeled on Tableau. To assess if Voyager helps users systematically explore more data, we analyze data field coverage. We found that subjects interacted with 1.5 times more unique field sets in Voyager. For user ratings (Figure 8), Voyager was preferred for open exploration as it gave users *"options that [they] wouldn't have thought about"*. However, users preferred PoleStar for question answering as they could build plots specific to their questions. Users also desired to *"start exploration with Voyager and switch to PoleStar to dive into questions"*. This result indicated the value of chart recommendation for open exploration, but also called for a unified tool that supports both manual authoring and recommendation browsing.

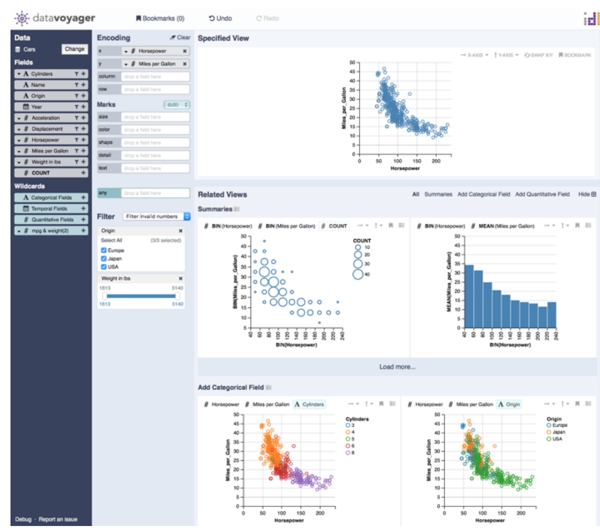

Figure 7 Voyager 2 blends manual and automated chart authoring. Users can use the shelf interfaces (left) to specify arbitrary views (top right). From the specified view, Voyager 2 presents related views, allow users to browse and discover relevant data fields and alternative ways to summarize or encode the data (lower right). Users can also use the *wildcard* interfaces (in teal) to specify multiple charts in parallel by varying certain chart properties.

Voyager 2: Blending Manual and Automated Chart Authoring

Motivated by the complementary value of manual chart authoring and recommendation browsing shown in the Voyager study, we designed **Voyager 2**, a tool that blends manual and automated chart authoring to facilitate *both* open exploration and question answering in one tool.

With Voyager 2 (Figure 7), users can *pivot among multiple interaction methods* within one system. As in traditional visualization tools like PoleStar and Tableau, users can *manually create arbitrary views* (Figure 7, top right). Moreover, Voyager 2 presents *two new partial view specification interfaces*. **Related views** suggest charts based on the current specified view, allowing users to browse charts with *relevant data fields or alternative ways to summarize or encode the data* (Figure 7, lower right). The **wildcard** interfaces enable analysts to *specify a set of charts in parallel* by varying chart properties, giving them control over sets of views aligned with their analysis goals. To transition from browsing to follow-on analysis, users can make any suggested views the new specified view, and modify the view or browse other relevant views.

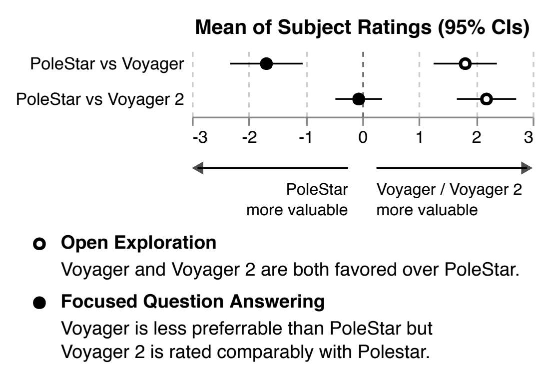

Figure 8: Task-based subject preference from studies comparing Voyager and Voyager 2 with the PoleStar specification tool. Voyager 2 is *overall* favored over Voyager and PoleStar for supporting *both* open exploration and focused question answering.

We evaluated Voyager 2 by comparing it with PoleStar. Akin to Voyager, Voyager 2 helped users systematically explore more data (2.4 times more unique field sets interacted with), and received higher ratings for open exploration (Figure 8). For question answering, while users preferred Polestar to Voyager, users rated Voyager 2 comparably to PoleStar. Despite having more features than PoleStar, users remarked that Voyager 2 was *"easier to use"* and *"more learner-friendly"*. Overall, Voyager 2 improved upon Voyager and PoleStar in terms of supporting *both* exploration and question answering. In an ongoing work with the Jupyter team, we are integrating Voyager 2 as a JupyterLab plugin, so users can easily explore data in Voyager 2 and transition from and to other analysis phases, such as cleaning data and sharing results, within the Jupyter platform.

Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow

Besides supporting data exploration, visualization is also a critical tool for understanding machine learning. To help developers understand deep learning architecture, I led the development of the TensorFlow graph visualizer [5], a tool that combines automatic layout techniques with user interaction to visualize dataflow graphs of TensorFlow models.

To simplify creation and deployment of deep learning models, Google's TensorFlow library generates low-level dataflow graphs to represent computations in the models. As developers often draw high-level diagrams to build a mental map and communicate their model structures, they desire a way to automatically generate diagrams from their code. However, these graphs are low-level and typically contain thousands of nodes. Some of these nodes also have high degrees but are unimportant for understanding model structure. As a result, standard graph layout tools produce tangled diagrams (Figure 9, top).

Combining automatic layout techniques and interactions that give users control, the visualizer enables developers to create legible interactive diagrams that match their mental maps. The tool applies hierarchical graph clustering to build a high-level diagram of the model (Figure 9-10), akin to what developers typically draw, and enables users to explore its nested structure by expanding clusters. To help users create graph clusters that match their mental maps, our strategy is to let users annotate the source code with hierarchical information. To declutter the layout, the tool applies heuristics to extract nodes that developers normally omit from their hand-drawn diagrams and allows users to customize the layout by extracting and un-extracting more nodes.

The graph visualizer has been released as a part of TensorFlow and widely used in the community for debugging and sharing their deep learning model structures. Online reviews of deep learning libraries mentioned that the visualization "*helps differentiate TensorFlow from other libraries"* and *"is a great step in the right direction"*. Screenshots from the visualizer regularly appear on the official and 3rd-party tutorials as well as on StackOverflow questions for explaining models. From internal mailing lists at Google and external blogs, we have found that many users repeatedly modified their annotations in the code to make the diagrams match their mental map, indicating that the visualizer is indeed valuable for model developers.

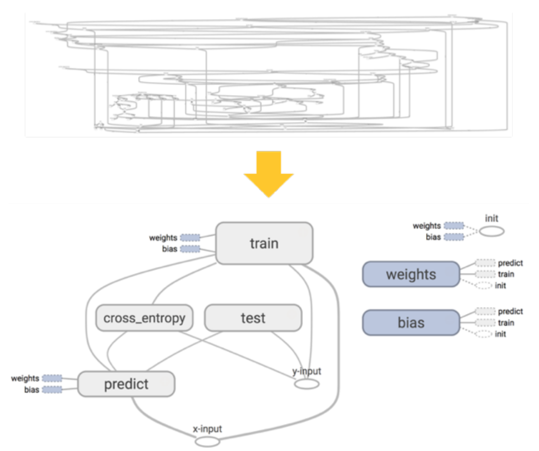

Figure 9. TensorFlow Graph Visualizer converts the low-level graph of a linear model ("Hello World" example) into a high-level interactive diagrams.

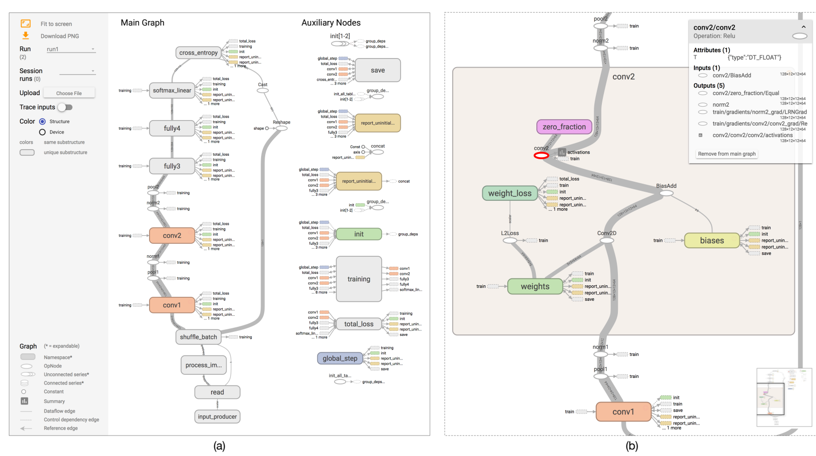

Figure 10. A convolutional network for classifying images. (Left) An overview shows a flow of nodes that are grouped based on user annotations. Unimportant nodes are extracted to the side. (Right) Expanding a group shows its nested structure.

Future Research

My mission is to help people work with and benefit more from data via visualization and intelligent systems. To make an impact on the company's products as well as internal and open-source tools, I plan to foster collaboration with designers, engineers, product managers, and other researchers. Through collaboration, I will identify vital data challenges in the company and address them by building and augmenting systems with automatic and interactive techniques. Here are some research challenges that excite me.

Improve foundations & design new applications for chart recommendation. My thesis work has contributed foundations for chart specification and recommendation, and used them to facilitate data exploration. However, this work is only an initial investigation of chart recommendation research. While CompassQL currently only suggests single-view plots, multi-view composition and interaction support in Vega-Lite can enable a richer set of recommendations. Layered charts such as box plots can provide more statistical information like variability and outliers in addition to the central tendency of a distribution. Suggesting interaction techniques for different types of plots can facilitate interactive analysis. Besides data exploration, automated chart design can assist other tasks. Suggesting annotations to indicate trends may enhance visual communication. Warning about misleading plots and proposing alternatives with explanations for why certain designs are better can educate novices and improve visualization literacy.

To support a richer set of recommendations, we need to study and extend enumeration strategies, design constraints, and rankings in CompassQL beyond single-view plots. For example, suggesting multi-view graphics requires enumeration of composition structures (e.g., by layering or concatenating views) in addition to enumerating wildcards in a fixed specification structure. A critical question is how to provide an intuitive mechanism for users to constrain the space of suggested composition structures. Built-in design constraints such as enforcing consistencies of visual encodings between sub-views [10] can also facilitate reading of multi-view graphics. Moreover, manually designing features for the perceptual effectiveness ranking may not scale for this large combinatorial design space, I plan to explore probabilistic models that learn and improve the rankings based on user interactions. Such work will also open up possibilities for personalization and adapting recommendation for different data domains.

Help people create and understand machine learning systems. Beyond depicting model structures from their source code, I am interested in designing tools to support visual authoring, exploration, and analysis of machine learning models. To support these tasks, many important questions remain. What is the right abstraction level for building blocks in the model authoring interfaces? What are variations of the models that developers expect to vary in their experiments? How might we design interfaces and visualizations to help people better explore and evaluate these variations? Besides supporting model creators, I would like to help end-users better understand models. For example, I am interested in designing visualizations to explain why interpretable models such as decision trees or regression models make certain recommendations or predictions.

Design domain-specific visualizations. I am also interested in visualization techniques for domain-specific data. For example, I would like to explore if we can apply our strategy from the TensorFlow graph visualization to extract unimportant nodes for other kinds of graphs such as dataflows of interactive visualizations in Vega. Moreover, I believe there are high-impact opportunities to help build visualization tools and techniques for exploring and understanding other domain-specific data including text (e.g., log and social interaction data), temporal sequence data (e.g., for funnel analysis), and image data. It is also interesting to explore if we can build a domain-specific language and build a recommender engine for visualizations in these specific domains.

Interaction design for intelligent systems. I am also intrigued by general design issues for intelligent systems, namely how to help end-users benefit from automation while preserving their control and creativity. Some insights from my work on visualization tools could be applicable for other domains. A vital design consideration is to find the right balance between automation and user control. For trivial decisions such as producing low-level chart components in Vega-Lite, simply automating the process while providing a way to override can be sufficient. For more complex activities like data exploration, it is better to let users pick from multiple suggestions and provide fine-grained controls over the suggestions like in Voyager 2. Partial specifications of domain-specific representations like CompassQL queries can facilitate such control over the suggestions. A related consideration is to identify parts of the work that need user input. Without user annotation, graph layout techniques in TensorFlow would not produce diagrams that match users' mental maps. Despite generally positive results for Voyager 2, a study participant remarked that *"[the related views] are so spoiling that I start thinking less."* Going forward, the bias and complacency due to automation is another important issue. Besides visualization, I am keen to apply these considerations to augmenting systems with automation in domains such as statistical modeling and web design. By studying and designing systems in diverse domains, I hope to generalize observed design patterns and develop guidelines for building systems that empower people with machine intelligence.

References

1. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, Jeffrey Heer*. IEEE TVCG (InfoVis) 2015. **Invited to SIGGRAPH 2016 as 1 of 4 top TVCG papers**.
2. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. *Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, Jeffrey Heer*. ACM SIGCHI 2017.
3. Vega-Lite: a Grammar of Interactive Graphics. *Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer*. IEEE TVCG (InfoVis) 2016. **Best Paper Award.**
4. Towards A General-Purpose Query Language for Visualization Recommendation. *Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, Jeffrey Heer*. ACM SIGMOD Human-in-the-Loop Data Analysis (HILDA) 2016
5. Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow. *Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mané, Doug Fritz, Fernanda Viégas, Martin Wattenberg*. IEEE TVCG (VAST) 2017. **Best Paper Award.**
6. Declarative Interaction Design for Data Visualization. *Arvind Satyanarayan, Kanit Wongsuphasawat, Jeffrey Heer*. ACM UIST 2014
7. GraphScape: A Model for Automated Reasoning about Visualization Similarity and Sequencing. *Younghoon Kim, Kanit Wongsuphasawat, Jessica Hullman, Jeffrey Heer*. ACM SIGCHI 2017. **Best Paper Honorable Mention.**
8. Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images. *Jorge Poco, Jeffrey Heer*. EuroVis 2017
9. Show me: Automatic presentation for visual analysis. *Jock Mackinlay, et al.* IEEE TVCG (Proc. Infovis) 2007
10. Keeping Multiple Views Consistent: Constraints, Validations, and Exceptions in Visualization Authoring. *Zening Qu et al.* IEEE TVCG (InfoVis) 2017
11. A Conversational Agent for Data Science. *Ethan Fast*. https://hackernoon.com/4ae300cdc220
12. SeeDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *Manasi Vartak, et al.* VLDB 2016
13. Making Data Visual: A Practical Guide to Using Visualization for Insight. *Miriah Meyer, Danyel Fisher*. O'Reilly Media 2018