A Declarative Specification for Clinical Reference Range Number Line Visualizations

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Computer Science,
Bioinformatics & Computational Biology

By:

Fisk, Zoe
Lin, Skyler
Medailleu, Zachary
Vasiliou, Morgan

Project Advisor:

Lane Harrison

Date: March 18th 2026

A Declarative Specification for Clinical Reference Range Number Line Visualizations

Zoe Fisk, Skyler Lin, Zachary Medailleu, and Morgan Vasiliou

Fig. 1: In the Clouds: Vancouver from Cypress Mountain. Note that the teaser may not be wider than the abstract block.

Abstract—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum. A free copy of this paper and all supplemental materials are available at `https://OSF.IO/2NBSG`.

Index Terms—Radiosity, global illumination, constant time

1 Author Details

You should specify ORCID IDs for each author (see `https://orcid.org/` to register) for disambiguation and long-term contact preservation. Use `\authororcid{Author Name}{0000-0000-0000-0000}` for each author, replacing the "Author Name" and using the 16-digit (hyphenated) ORCID ID for the second parameter. The template shows an example without ORCID IDs for two of the authors. ORCID IDs should be provided in all cases.

Each author's affiliations have to be provided in the author footer on the bottom-left corner of the first page. It is permitted to merge two or more people from the same institution as long as they are shown in the same order as in the overall author sequence on the top of the first page. For example, if authors A, B, C, and D are from institutions 1, 2, 1, and 2, respectively, then it is ok to use 2 bullets as follows:

- A and C are with Institution 1. E-mail: {a | c}@i1.com .
- B and D are with Institution 2. E-mail: {b | d}@i2.org .

- *Zoe Fisk is with Brown University. E-mail: zfisk@example.com*
- *Skyler Lin is with Grimley Widgets, Inc. E-mail: ed.grimley@example.com.*
- *Zachary Medailleu is with Martha Stewart Enterprises at Microsoft Research. E-mail: martha.stewart@example.com.*
- *Morgan Vasiliou is with Martha Stewart Enterprises at Microsoft Research. E-mail: martha.stewart@example.com.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

2 Introduction

The volume and availability of healthcare data have dramatically increased in recent years with the introduction of personal health devices and patient portals[30, 8]. Personal health data can inform patients and guide health-related decisions, but low literacy levels can inversely affect their potential[27]. 21% of Americans have low literacy skills, and 36% of American adults have basic or below basic health literacy, with disadvantaged populations disproportionately making up these groups[11, 16]. The stress of retrieving healthcare information, especially in emergency or intensive care situations, is an additional factor in decision-making. Stressful situations can increase cognitive load, negatively impact information retention and cost-benefit analysis, and lead to habitual decision-making.[25]. Literacy rates and stressful situations make it difficult to understand, retain, and apply health information towards decision-making, but data visualizations can help alleviate this by conveying complex information and reducing cognitive load[24].

Reference range number lines (RRNLs) are visualizations with promising results for increasing comprehension of health information[5, 6, 18, 21]. These visualizations display a single-value datum along a continuous scale that is categorically classified based on reference ranges for that metric, such as the exact value of a blood glucose test result being indicated on a number line and also classified as low, normal, or high based on the reference ranges [18, 21]. However, RRNLs specifically are not often studied by health researchers. In some cases,

researchers study visualizations that share common attributes with reference range number lines, but do not follow the exact same format, such as horizontal bar charts for lab results or number lines for understanding risk[9, 12]. Generalizing findings about RRNLs and similar visualization types is difficult because RRNLs are not well-defined or formalized. As a result, there is no current formalized style guide or method for designing RRNLs.

Computational grammars and specification languages can standardize RRNLs by translating their basic components into a framework to support systematic replication[19]. RRNLs are tailored with individualized data and different color schemes, but the basic structure of the visualization stays the same[4]. A grammar allows for the replicability of the basic structure and also the expressiveness to tailor the visualization as needed. Grammars also enhance accessibility by helping computers parse visualizations and generate accurate summaries. These summaries could then be translated into other languages or improve accessibility for blind individuals. Exploring current grammars and their support for RRNLs is a vital first step to increasing accessibility and formalizing RRNLs.

In this paper, we analyze the high-level grammar Vega through the lens of reference range number lines replication and create a basic framework for replicating RRNLs. The background ties together the current knowledge of RRNLs, health visualization techniques from the visualization side, and existing grammars. Our methods detail our annotations of RRNL examples and visualization features, and how we translated those into properties for a grammar. We then note how we replicated these RRNLs in Vega and the current limitations of such methods. We then use our grammar to replicate a RRNL study. Our discussion summarizes the success in replicating RRNL properties and gives code for RRNLs with different properties. These results are also presented on a web page for future reference. We conclude with a limitations section that notes the limitations of our grammar and future considerations for expanding it to fit the needs of RRNLs, with the ultimate goal of giving RRNLs a space in the visualization community and improving accessibility.

[Sum-up of contributions]

3 Background

3.1 Reference Range Number Lines

Reference range number lines (RRNLs) are used in high-impact scenarios for patients with chronic diseases[18, 21, 20, 29, 9, 26, 10]. They serve as tools for chronic disease management by demonstrating the patient's current health status based on lab reports and self-reported surveys. For patients with greater barriers to comprehending their health status, easily interpretable graphs make the difference between good health management and adverse effects. Reference range number lines have been shown to improve comprehension for individuals with low health literacy and older adults, both of whom have historically higher rates of chronic illness and barriers to comprehension. By exploring how RRNL design variations improve comprehension, researchers can inform graphical design to ensure high-risk individuals stay in control of their health.

Recent RRNL studies explore user preference for RRNLs and how design variations influence comprehension, urgency and risk perception, and self-management behaviors. RRNLs are strongly preferred by users compared to other visualization types. For example, Arcia et al. (2019) found that RRNLs were preferred over a stoplight design for Asthma Control Status [5] while Brewer et al. (2012) established a preference for RRNLs over a tabular data format displaying lab results [9]. Further research has gone into the specifics of designing easily comprehended RRNLs. Arcia et al. (2016) [7] employed iterative participatory design methods to create infographic designs, including RRNLs, for low health literacy individuals. They concluded that successful infographics are information-rich and provide context. This finding became a foundation for other studies that measured how additional contextual cues influence user comprehension. One study compared four RRNL designs with increasing contextual cues by adding reference range color, clear reference range titles, and visual references to average values for a demographic to each subsequent design [18]. They found that adding contextual cues does not increase cognitive load and could therefore be included in RRNLs to increase the information richness of a design. Another study found that adding colored anthropomorphic cues, as compared to black-and-white designs with and without anthropomorphic cues, decreased task completion time and increased comprehension of health data [28], further demonstrating the value of contextual cues in RRNLs. Outside of adding contextual cues, other studies have looked into the relationship between altering cues and the resulting impacts on user performance. For example, Zikmund-Fisher et al. (2017) [31] explored how variations in color for reference ranges influence perceived urgency and desire to contact health care providers; they found that gradient coloring, as compared to block, resulted in greater sensitivity to variations in test results. These findings create a baseline for information-rich visualizations built to support health data comprehension and informed decision-making. However, the current methodologies for creating reference range number lines are not systematic, and the computational tools used are often omitted.

3.2 Health Visualizations

Computer visualizations are often used in health to improve risk communication and Bayesian reasoning for medical decision-making. Current visualization tools have implemented individualized visualizations to tailor the decision-making experience to the patient. For example, Linhares et al. proposed a visualization tool called ClinicalPath that displays a longitudinal map of a patient's clinical data using symbols and color encoding [17]. This tool used reference ranges for test results and displayed the classification of a test result into these reference ranges by encoding them with symbols. Other visualizations in ClinicalPath included line charts with test results over time, showing both single tests and how they compared to other tests over time. ClinicalPath was received positively by participants and produced strong accuracy scores from participants. Another visualization tool, PROACT, was developed to communicate prostate cancer health risk[14]. PROACT employed pie charts, bar graphs, and temporal area charts to showcase different dimensions of risk. Both doctors and patients found this tool to be useful, but some doctors recommended changing the sequence of visualizations to fit the narrative sequence they follow when having patient conversations. Additionally, some participants showcased that they were unfamiliar with or had difficulty understanding the temporal area charts. Current computer visualization tools are working to enhance risk perception and continue to inform their design with user feedback, a part of which relies on the ability of a user to understand a graph type.

While bar charts and pie graphs are often used to communicate risk, icon arrays (also known as pictographs) also have a dedicated role in displaying health risk and are a well-studied visualization in the health and visualization communities[14, 3, 2]. Icon arrays are graphics that depict numerators and denominators of a probability using a matrix of icons, such as human figures[3]. When directly compared, these graphics have been shown to outperform other visual forms of risk communication, like bar graphs[15]. They also have the potential to lower barriers to patient understanding of probability and ratios, such as denominator neglect, where a person only sees the numerator of a probability instead of the numerator and denominator[13]. However, the efficacy of icon arrays has been previously determined to be influenced by individual measures. For example, Okan et al. [22] explored how graph literacy impacts the efficacy of icon arrays for overcoming denominator neglect and found that icon arrays more often increased risk comprehension among individuals with high graph literacy as compared to those with low graph literacy. Another study by Ottley et al. [23] measured the impacts of spatial ability on icon arrays and Bayesian reasoning. Ottley et al. tested a variety of text and visualization conditions to demonstrate Bayesian reasoning and used icon arrays as the main visualization type. After analyzing participants' responses, they found that participants with high spatial ability performed significantly better than participants with low spatial reasoning across all conditions. These studies showcase a small fraction of the work going into health visualization that, when aggregated, will inform strong designs that enhance risk perception and patient comprehension.

3.3 Grammars

The first step we took to address this gap successfully was turning to visualization design problems on specifically reference range number lines that have been approached in other contexts; we examined domain-specific frameworks that support the creation of complex graphics. Systems such as EnTICE3 and BITSI focused on having streamlined design decisions that enabled visual outputs using formalized style guides. This offered a lot of valuable insight into how semantic intent is preserved through user implementations. While these framework/systems are not traditional visualization grammars with foundational levels, they showcase the importance of representation of interpretive elements.

There are current frameworks that support the creation of complex, tailored graphics. Although they are not traditional visualization grammars, they showcase the importance of the representation of interpretive elements. One notable example is EnTICE3 (Electronic Tailored Infographics for Community Engagement, Education, and Empowerment), a system developed to support the generation of customizable infographics from personal health data. EnTICE3 uses a special style guide as the main communication tool between domain experts and software developers, which ensures that design specifications are recorded consistently and can be interpreted easily. The style guide is made up of standardized fields that describe graphical attributes of each infographic. Consistency and clarity across resulting visualizations show the possibility that a structured approach, like EnTICE3, can clearly communicate design specifications across disciplines, even without prescribed specific implementation details. EnTICE3 also emphasizes semantic transparency in the health communication spaces. While EnTICE3 does not define a visualization grammar, the style guides and framework with clear explanations for each have helped inform our thinking immensely around reference range number lines.

Another structured framework that informed our thinking on visual design contexts was BITSI (Browser-based Infographic Tailoring Self-service Interface), which, similar to EnTICE3, focuses on supporting the creation of health visualizations through well-defined design structures. Even though BITSI isn't fully a visualization grammar, it emphasizes the intentional design of sensitive health information in a clear and direct manner through a formal style guide. A few examples of the semantic design decisions that were encoded were how variables were displayed, color mapping, and text placement; all of these elements are tweaked to perfection, ensuring that tailored infographics remain consistent with designs that were validated with interviewed patients.

A study featured in the paper showed the influence of BITSI's structure through the visualization of Asthma Control Test (ACT) scores using a number line format. The BITSI framework generated individualized infographics that displayed either one to two ACT scores over time, with clear, well defined thresholds that communicated levels of asthma control. Through BITSI, these interpretations were encoded directly into the system logic and style guide; this saved notable runtime for designers who would likely need to interpret this if the thresholds were not systematically encoded. Much like EnTICE3, BITSI separates what information is being communicated from how it is shown visually to users. For example, BITSI approached this through a browser-based, point and click interface that eliminated the need for users with surface level access to edit templates with confusing syntax. Instead of having clinicians and researchers without coding skills interacting with coding interfaces, BITSI's browser based system allowed users to easily input patient data, preview results, and generate finalized PDFs. This design decision reduced the potential for user error through tailoring these constraints and exhibited the balance between control and usability. Overall, BITSI's design was another strong example of the importance of preserving semantic intent and handling health data on number lines with formalized design specs.

Using these ideas within the two frameworks, we transitioned into studying grammar-based visualization systems; a more structured and declarative approach to specifying visual designs and features we wanted to include in our own grammar. Rather than relying on fixed chart templates pulled from hundreds of different sources, these three structured grammars stood out to us because of their formalized data-driven decisions and creative visual components. Below, we have summarized several grammar-based systems that informed our thinking and directly influenced the development of our own descriptive language for reference range number lines. Each of these works illustrates how overlooked specific visual elements can be elevated into bolder constructs within a visualization grammar.

In GoFish: A Grammar of More Graphics!, Pollock and Satyarayan highlight limitations in the traditional Grammar of Graphics, which is the framework that underlies many common visualization tools, such as Vega-Lite. The authors argue that, while Grammar of Graphics has enabled more expressive design beyond fixed chart types, there are still some gaps when attempting to represent less conventional visual forms, which forces designers to rely on custom, low-level graphics. To address these problems, they propose GoFish, a new declarative grammar that uses primitive shapes and graphical operators based on Gestalt to build more complex visualizations. This work illustrates the need to reconsider which visualization types are recognized and taught within visualization literacy frameworks. By prioritizing only familiar charts such as bar charts and line graphs, educational models risk overlooking other forms, such as number lines, that may be equally valuable for improving comprehension.

AnnoGram: An Annotative Grammar of Graphics Extension proposes an extension to Wilkinson's Grammar of Graphics that treats annotations as priority components of a visualization over manual overlays. In many other existing visualization tools, annotations like labels, markers, reference indicators are added using very basic graphic elements. This approach to developing annotations makes it difficult to recycle and reuse elements across different, diverse visualizations. What stood out to us in AnnoGram is its emphasis on semantic intent over simple visual implementation. In the paper, AnnoGram introduces concepts from annotation targets to placement strategies that actually allow designers to specify what should be annotated and give a solid reasoning, rather than just a manual encode on how annotations should appear in the given pixel space. This declarative approach aligns closely with our goals regarding our reference range number line grammar, where elements like value indicators, reference ranges, and even specific threshold metrics will carry valuable semantic meaning that should remain consistent across designs.

AnnoGram's approach also closely parallels ideas introduced in GoFish, the first grammar-based system we drew inspiration from. In particular, its emphasis goes beyond just static chart types and low-level encodes and reaches to more composed building blocks. While GoFish focuses on constructing complex visualizations, AnnoGram focuses on formalizing annotations for said visualizations. Both frameworks had great principles that helped to expand the expressive capacity of visualization grammar.

4 Methods

4.1 Number lines

To create our grammar, we delved into the basic components of reference range number lines and translated them into a grammar and the resulting Vega objects. We then tested the capabilities of our grammar by replicating a RRNL study and reproducing the visual stimuli with our creation. We used d3 to compensate for any limitations of our grammar that were outside of the scope of Vega.

4.1.1 Research into number lines

As data visualization continues to expand in complexity and application, a range of software and libraries have been developed to support the creation of visual displays, including Vega-Lite and D3. Vega-Lite is a grammar that uses an intuitive JSON syntax to allow the creation and customization of many types of visualizations (source). It is based on Vega, a similar grammar, that offers similar functionality with a greater degree of versatility than Vega-Lite. D3 is a Javascript library. It is a lot more versatile than Vega-Lite or Vega, but comes at the cost of complexity (source). However, none of these libraries include a dedicated method for constructing number lines. Instead, we would utilize their versatility to design RRNLs using built-in features. To start

with our development of a specification language for creating RRNLs, we chose Vega-Lite. By starting with the more concise choice of the languages, we could build prototype visualizations more efficiently and measure the capabilities and limitations of Vega-Lite to create RRNLs. After identifying some notable limitations, we based a second iteration of our specification on Vega to generate more accurate RRNLs. We then used D3 to fill in a gap that wasn't feasible from our understand of Vega's capabilities.

4.1.2 Identifying Reference Range Number Line Components

To begin our process, we needed to locate RRNL examples in the current literature. We searched three databases for studies containing RRNL examples to annotate. We used PubMed, IEEE, and ACM, and used search terms such as "'reference range' AND 'number line'" or "'number line' AND 'graph.'" The search terms were iterated on to ensure we found all possible RRNL examples, since there is no standardized way to refer to these graphics.

The first step in creating a grammar was identifying the components that make up RRNLs. By highlighting the components, we can identify common trends in RRNL design to build a baseline structure and identify levels of customization. We researched health-related papers that use RRNLs to visualize data and annotated their number lines based on the components making them up. Our annotations followed a code that we iterated as needed to include different RRNL examples.

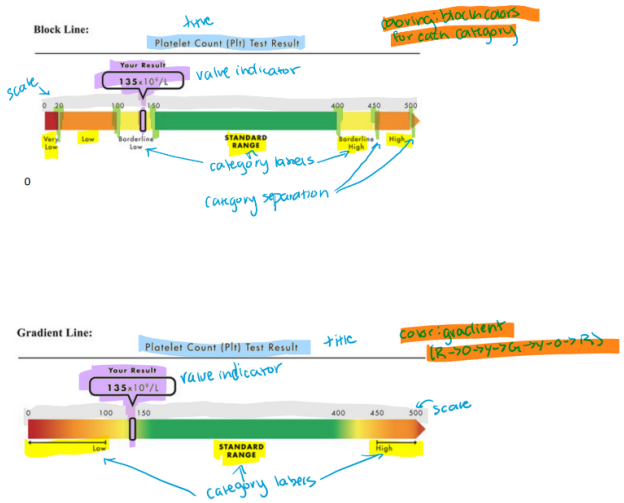

Fig. 2: Annotated Zikmund-Fisher Platelet Count RRNL

The finalized taxonomy is as follows:

Component Name	Classification Requirements
Title	Phrase or sentence introducing or labeling the accompanying visualization. Subtitles or individual labels for each RRNL in double RRNLs were included in this category.

4.1.3 Number Line Specification Language Properties

Using the components identified in our annotated number lines, we created documentation for the properties an object in our language would require. With each component of a RRNL, we needed to consider how that property may be represented in a language and then determine what additional properties would be necessary to allow for enough customization to replicate many variants of RRNLs with the same language. For example, with these properties, it can be specified where the position of the indicator is, what the title text of it is, and whether it overlaps the RRNL or points to it. These properties aren't exhaustive as to how RRNLs can be constructed, and instead represent the common components found in RRNLs to allow our language to create them without too much complexity.

We also chose to represent the reference ranges as a list of ordered categories, each of which contains the name of the category, the color representing it on the number line, and the position the number line ends at. We made the decision for each category to store only its end value because the "data" property stores where the first category starts, and each subsequent category will immediately start after the previous one ends. Formatting the language this way prevents users from inputting the same values twice, therefore reducing error in data entry. With our list of properties, we created several objects using this specification language based on our annotated RRNL examples. Each property was set to match the RRNLs so that one could recreate these number lines as closely as possible given just the objects.

4.1.4 Capabilities of Creating Number Lines in Vega-Lite

While the properties we defined could theoretically be used to construct RRNLs comparable to the examples, we did not yet have a library that could use these objects to recreate RRNLs with the same data, components, and customization. Instead, we used Vega-Lite to build visualizations that resembled the number lines as closely as possible. By creating a RRNL in Vega-Lite using the same properties from an object from our specification language, we could examine how viable it would be to use the grammar for generating RRNLs.

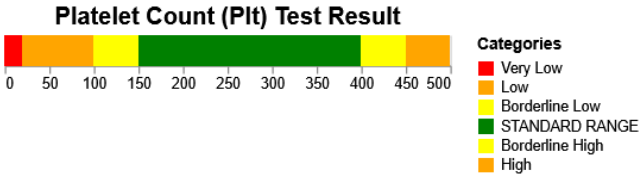

Fig. 3: Zikmund-Fisher Platelet Count RRNL Replicated in Vega-Lite

Since there is no true number line chart type in Vega-Lite, the best way to work around this limitation was to build it as a horizontal stacked bar chart with no Y-axis. This would draw only a single row and create a visualization similar to a number line. The width of the line is extended to mimic a bar, and the X-axis title is removed so that only the number line title remains. Each separate category in the horizontal bar chart becomes its own value in the data with a unique name and color. This way, the graph is recognizable as a RRNL even though it is not technically implemented as one. The title and tick marks were already existing components within Vega-Lite, so they were straightforward to incorporate. Getting the categories to reflect the correct sizes and appear in the right order required additional adjustments, as the categories in our specification language each have an end value property, but the end value is not the same as the size of the category on the graph. To ensure the order and category sizes are correct, each category is instead given a property based on the range it spans and an additional property to set the order of the categories. By using Vega-Lite's order functionality on this new property, the categories correctly span their intended ranges in the same way as the example RRNL.

However, there are several limitations to creating number lines in Vega-Lite, as some of the components and features of RRNLs are not supported in the grammar. These issues stem from the fact that we were not technically working with number lines, but rather stacked bar charts. From our understanding of Vega-Lite's capabilities, some components of RRNLs that aren't typically found in bar charts are difficult or unfeasible to implement in Vega-Lite. The most pertinent example is the value indicator, which is an imperative component of RRNLs that displays to the user precisely where a specific point of data falls on the number line for simple comprehension. Similarly, there is currently no way to include category separators in the number line. Many reference range number lines use gradients instead of blocks of color, but Vega-Lite has no standard implementation of adding a gradient to bar charts. Other features of RRNLs could be implemented, but with limits on customization. For example, the category labels of

RRNLs are usually positioned underneath or on top of their respective categories, but in Vega-Lite, they are restricted to a legend to the side of the graph. Some RRNLs also only have tick marks at the category separators, but in Vega-Lite, tick marks will always be evenly spaced out.

4.1.5 Conversion to Vega-Lite Objects

To demonstrate that our specification language could be used to construct RRNLs, we created a program that takes in an object representing a RRNL in our specification language and outputs a Vega-Lite compatible object. We used Python as the language to implement this process, as it is a useful language for automating similar tasks. The program takes in a JSON file as input that is configured with appropriate properties from our specification language. The program then reads each property into a Python object, and these properties are used to construct a dictionary with key value pairs in the same format as a Vega-Lite object. Properties of components that aren't supported in Vega-Lite, such as value indicators and category separators, have no impact on the output. When the dictionary is complete, it's converted to a JSON object and output to a file by the program. This file can be opened in Vega-Lite to construct a RRNL with the same data and properties from the original JSON file.

This program allows a user to conveniently create a RRNL in Vega with their own data. The input files using our specification language take in properties that are important to the creator of the number line, such as the title positioning, the names of categories, and the number of tick marks. Then the program uses these properties to create a readable RRNL in Vega-Lite. Because RRNLs aren't officially supported in Vega-Lite, a user might not know how to construct one. It is not immediately apparent that you can imitate one with a stacked bar chart, and figuring out how to position the categories correctly might prove difficult. This program handles that part for the user, meaning that they are only responsible for inputting the data and the type of customizations they want. Although not all of the components are supported in this program, it demonstrates the possibility of a more straightforward and convenient way for users to construct RRNLs.

4.1.6 Conversion to Vega Objects

Number lines aren't officially supported in Vega, but Vega's expressiveness allows for closer replication of example RRNLs by supporting additional features that Vega-Lite could not replicate. When creating RRNLs in Vega, the process of creating the base number line is largely the same as it is in Vega-Lite, and the primary changes are the additional components that couldn't be implemented with Vega-Lite alone.

The two features of Vega that were most useful in creating more accurate RRNLs were the ability to set a starting and ending X-axis value for each bar and the ability to draw text and symbols onto the visualization with exact coordinates. Each bar of the number line will get a field for the start and end values of that bar, meaning the graph can use these values to draw components at the exact positions that they're expected. The mark functionality of Vega allows for symbols, lines, or text to be drawn onto the visualization. They can be set to be drawn at exact positions, meaning that with the right values for the value indicator's text, arrow head, and arrow body, a value indicator can be created in Vega even though it's not an intended Vega feature. The data from the user's object made in our grammar is integrated into the relevant position values, and the final outputted Vega object will draw the category separators, category labels, and value indicator in their correct positions with minimal input from the user.

Similar to the Vega-Lite code, we created a program that will take a file using our specification language and generate a Vega object that will create a RRNL using the properties of the object. It works the same way as before, except now it takes into account the properties that couldn't be implemented in Vega-Lite and uses them to create a more accurate RRNL. Additionally, it will generate an HTML file with the Vega object embedded. The program supports the input of an array of objects in our grammar as well. Each Vega object will be output to its own JSON file, and they'll all be present in the outputted HTML file.

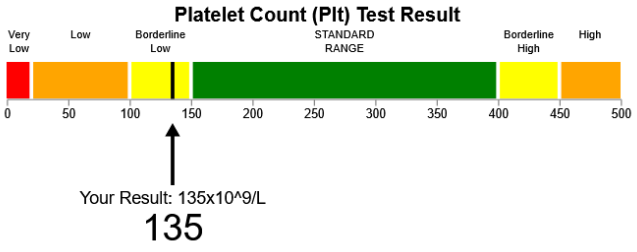

Fig. 4: Zikmund-Fisher Platelet Count RRNL Replicated in Vega

4.1.7 Gradients in D3

The only feature that is shared between many RRNLs that Vega didn't have a clear solution for is the presence of a gradient across the number line instead of distinct block coloring based on category. Because of how common gradients are and how they make an important difference to how one perceives a RRNL[31], we aimed to find an alternate way for users to easily add gradients to their number lines. Due to D3's versatility, we chose to use it as a library in order to generate number lines with gradients. Since Vega allows exporting objects as SVGs and D3 allows importing SVGs into HTML files, we could use the visualization that was already created in Vega and add a gradient to it.

The program we created will take in an svg file created from a Vega object and locate the position and color information of each bar of the number line. It then removes the separate bars that are used to draw the RRNL and replaces it with a single bar of the same size. The information of each bar is used to create a gradient, and the new bar is filled in with this gradient. The resulting page from this file will display the inputted RRNL with a gradient added to the number line. With a combination of our Vega and D3 code, a user can create a RRNL with a gradient by only inputting some necessary parameters into our grammar and running it through our programs.

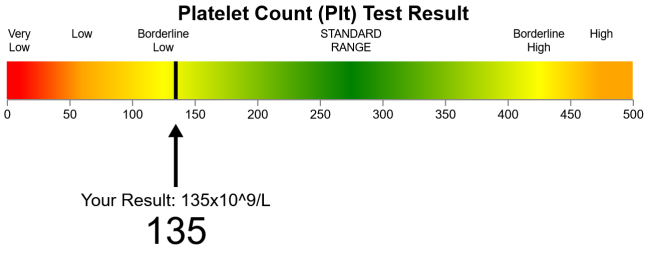

Fig. 5: Zikmund-Fisher Platelet Count RRNL Replicated in Vega with a gradient added via D3

4.2 Replicating a study

We decided to replicate a RRNL study with RRNLs generated from our grammar to see how well they did reproducing already established results. We chose to replicate a Zikmund-Fisher et al. study entitled "Graphics help patients distinguish between urgent and non-urgent deviations in laboratory test results." [31] This study was favorable because we were able to replicate the full scope of the methodology with only small changes to the tools used for participant recruitment and stimuli creation. Additionally, the range of visual stimuli in this study allowed us to find limitations in our grammar.

In the original study, they recruited participants through Survey Sampling International (SSI) and had participants complete the study on Qualtrics. Our participants were recruited through Prolific[1] and completed the study on ReVisit. ReVisit is a study platform designed to help researchers quickly develop and monitor their studies. Prolific is an online service that connects participants to studies based on criteria

set by the researcher. Participants are financially compensated for their time through Prolific.

The original study has four different visualization types, one of which is a table, while the other three are RRNLs with varying coloring schemes. We replicated the table using D3 and replicated the RRNLs with our DSL. We analyzed the original RRNL stimuli and created JSON objects with the accompanying information. For RRNLs with block coloring, we generated the RRNL in Vega using its accompanying JSON object and our preestablished Python script. RRNL JSONs with gradient coloring (i.e., simple gradient and colored gradient conditions) were generated with Vega code that was then saved to an SVG and added to our D3 script so we could match the gradient color scheme. To ensure that our generated HTMLs matched the composition of the original RRNLs, we had to make some small changes to the HTML that were outside the scope of our grammar. These changes included altering the arrow for our value indicator, so it pointed down at the number line instead of up, adding units to the test result, and selecting where to place our tick marks. The final visualizations were displayed to the users on an HTML webpage wrapped with ReVisit.

(Maybe a good place to put images of the number line generated w/ no additional meddling and then an image after meddling)

4.3 Website

The website we've been creating

5 Results

(put stuff in from replicated study)

6 Discussion

With how prevalent RRNLs are in the healthcare landscape, yet how little they're represented in visualization creation tools, we found it important to develop a method for users who aren't technically savvy to create RRNLs. With only a few properties in our grammar, a user can output a usable RRNL with all the necessary features of one.

While creating a RRNL with Vega was inherently complex due to the fact that they're not officially supported, we were able to complete our program by carefully deliberating over what is most necessary for RRNLs and how to reproduce them in Vega. Our final program takes in our understandable grammar and uses it to create a Vega object using the specified properties. Previously, creating a number line in Vega would require knowledge of JSON formats and indirect manipulation of other existing graph types to visually mimic RRNL formats. Creating an object in our grammar cuts out the time spent manipulating visuals and streamlines the process of making RRNLS, meaning that we've made it more efficient to create RRNLs from just some data points.

6.1 Accessibility and AI Implementation

(discuss how our DSL contributes to this idea, can reference the UpSet plots paper) Formalizing a visualization type through a declarative specification works toward our goal of creating accessibility. When the structure of a visualization like ours is encoded in a machine-readable grammar, it becomes possible for systems to describe that visualization without human intervention. Our DSL features the semantic components of RRNLs, such as reference range boundaries, category labels, and value indicators, and even named properties rather than as raw graphical primitives.

This structured representation creates an opportunity for screen readers, text-to-speech tools, and other tech to create accurate and consistent natural language descriptions of a patient's lab result directly from the specified object. This approach goes hand in hand with accessibility work done in other visualization domains. For example, efforts to formalize UpSet plots showed that a well-defined grammar standardizes creation and goes even further to enable the generation of textual summaries that can be used by assistive tools. Because the grammar is coded with intent, an automated system does not need to infer what a colored bar represents. It already knows that bar is a reference range category with a defined clinical meaning. RRNLs aren't used greatly with populations that have lower health and graph literacy, making this kind of output consequential for the exact audiences these visualizations are designed to serve.

6.2 Reducing errors and motivating researchers to do more of these studies

The DSL that we developed helps to reduce errors when creating reference range number lines, as it replaces the process of manually constructing them with a structured and reusable specification of the visualization's core components. Rebuilding each number line from scratch risks inconsistencies in category ordering, indicator position, label placement, or range values. Instead, the DSL can be used to create a consistent graph by defining the elements needed through a uniform set of properties.

This DSL is especially valuable in health communication research, as even small changes can threaten the validity of the study. Minor visual differences between graphs can result in drastically different interpretations. As such, we stress the importance of using consistent values and visuals. In addition, the DSL makes it easier for researchers to create graphs more quickly by reducing the barrier to creating RRNL stimuli, regardless of their visualization programming background. It also becomes more beginner-friendly to researchers without an extensive visualization programming background. By making RRNLs easier and simpler to generate consistently, our DSL can support stronger replication practices and encourage more researchers to conduct studies involving these visualizations.

7 Limitations

While our program is able to generate RRNLs using our grammar, it isn't able to create RRNLs with specialized details. Our grammar allows customization to fit a wide variety of general use cases, letting users customize aspects of the graph such as the color of each category, the position of the category labels, and the presence of components like category separators and value indicators. However, some examples of RRNLs have embellishments such as accompanying images for each category or alternate designs for value indicators. While these are possible in Vega and D3, we were limited by creating code that can create the type of graph a user wants straight from a grammar. Images in RRNLs vary in size, amount, and position, and accommodating all of these in our program while still keeping it convenient for users wasn't feasible with how our program is structured. A primary strength of our grammar is how accessible it is for anyone that wants to create a RRNL. While visualization libraries don't always offer an easy way to create a RRNL, our program will generate a RRNL in Vega with little input from the user despite the fact that they aren't officially supported in Vega. Since the scales and positions of images would likely differ depending on a user's needs, implementing support for image uploads would be counterintuitive to our grammar's ease of use. Despite these limitations, users who want a visualization that's more specialized can edit the outputted Vega object to add or adjust components in ways that aren't possible with our grammar. Many customized components, like value indicator design, would be possible to accomplish in D3 but would require creating code that goes beyond the capabilities of our grammar.

8 References

An example of the reference formatting is provided in the **References** section at the end.

8.1 Include DOIs

All references which have a DOI should have it included in the bibTEX for the style to display. The DOI can be entered with or without the `https://doi.org/` prefix.

8.2 Narrow DOI option

The `-narrow` versions of the bibliography style use the font `PTSansNarrow-TLF` for typesetting the DOIs in a compact way. This font needs to be available on your LATEX system. It is part of the `paratype` package, and many distributions (such as MikTeX) have it automatically installed. If you do not have this package yet and

want to use a `-narrow` bibliography style then use your LaTeX system's package installer to add it. If this is not possible you can also revert to the respective bibliography styles without the `-narrow` in the file name. DVI-based processes to compile the template apparently cannot handle the different font so, by default, the template file uses the `abbrv-doi` bibliography style.

8.3 Disabling hyperlinks

To avoid adding hyperlinks to the references (the default) you can use `\bibliographystyle{abbrv-doi}` instead of `\bibliographystyle{abbrv-doi-hyperref}`. By default, the DOI field in a bibTeX entry is turned into a hyperlink.

See the examples in the bibTeX file and the bibliography at the end of this template.

8.4 Guidelines for bibTeX

- All bibliographic entries should be sorted alphabetically by the last name of the first author. This LaTeX/bibTeX template takes care of this sorting automatically.
- Merge multiple references into one; e. g., use [**Max1995OpticalModelsDirect**, **Kitware2003VisualizationToolkitUsers**] (not [**Kitware2003VisualizationToolkitUsers**][**Max1995OpticalModelsDirect**]). Within each set of multiple references, the references should be sorted in ascending order. This LaTeX/bibTeX template takes care of both the merging and the sorting automatically.
- Verify all data obtained from digital libraries, even ACM's DL and IEEE Xplore etc. are sometimes wrong or incomplete.
- Do not trust bibliographic data from other services such as Mendeley.com, Google Scholar, or similar; these are even more likely to be incorrect or incomplete.
- Articles in journal—items to include:
  - author names
  - title
  - journal name
  - year
  - volume
  - number
  - month of publication as variable name (i.e., {jan} for January, etc.; month ranges using {jan #{/}# feb} or {jan #{--}# feb})
  - series. E.g., "TVCG", "TVCG/VIS" for special issue VIS papers, "EuroVis", "CGF/EuroVis".
- Use journal names in proper style: correct: "IEEE Transactions on Visualization and Computer Graphics", incorrect: "Visualization and Computer Graphics, IEEE Transactions on"
- Papers in proceedings—items to include:
  - author names
  - title
  - abbreviated proceedings name: e.g., "Proc.\ CONF_ACRONYNM" without the year; example: "Proc.\ CHI", "Proc.\ 3DUI", "Proc.\ Eurographics", "Proc.\ EuroVis"
  - year
  - series. E.g., "VIS" and "EuroVis" for short papers, "CHI"...
- Article/paper title convention: refrain from using curly brackets, except for acronyms/proper names/words following dashes/question marks etc.; example:

  The paper "Marching Cubes: A High Resolution 3D Surface Construction Algorithm" should be entered as "{M}arching {C}ubes: A High Resolution {3D} Surface Construction Algorithm" or "{M}arching {C}ubes: A high resolution {3D} surface construction algorithm". It will then be typeset as "Marching Cubes: A high resolution 3D surface construction algorithm".
- For all entries:
  - DOI can be entered in the DOI field as plain DOI number or as DOI url.
  - "pages" or "articleno": Provide full page ranges AA--BB, OR, if an article number is available like recent ACM conferences, use that instead. E.g., see the entry for Panavas et al. [**Panavas2022JuvenileGraphicalPerception**].
- When citing references, do not use the reference as a sentence object; e.g., wrong: "In [**Lorensen1987MarchingCubesHigh**] the authors describe ...", correct: "Lorensen and Cline [**Lorensen1987MarchingCubesHigh**] describe ..."

9 Appendices

Appendices can be specified using `\appendix`. For example, our Troubleshooting instructions in Sec. B.

Note that the paper submission has to end after the **References** section and within the page limit of the conference you are submitting to. Any version of Appendices or the paper with Appendices included has to be submitted separately as supplementary material. You can use the `hideappendix` class option to remove everything after `\appendix`. We encourage you to submit a full version of your paper to a preprint server with any appendices included.

You can use the `\iflabelexists` macro to cross reference an appendix from the main text, but only if that label (i.e. the appendix) actually exists. For example, above we use

```
\iflabelexists{appendix:troubleshooting}
  {\cref{appendix:troubleshooting}}
  {the appendix of the full paper at
   \url{https://osf.io/XXXXX}}.
```

in order to cross-reference to the appendix with `\cref` if it exists, but if the appendix is commented out then we will simply create a hyperlinked URL to it.

Supplemental Materials

Refer to the instructions for this section (**??**). Below is an example you can follow that includes the actual supplemental material for this template:

All supplemental materials are available on OSF at https://doi.org/10.17605/OSF.IO/2NBSG, released under a CC BY 4.0 license. In particular, they include (1) Excel files containing the data for and analyses for creating **??** and **??**, (2) figure images in multiple formats, and (3) a full version of this paper with all appendices. Our other code is intellectual property of a corporation—Starbucks Research—and there is no feasible way to share it publicly.

Figure Credits

Refer to the instructions for this section (**??**). Here are the actual figure credits for this template:

Figure 1 image credit: Scott Miller / Special to the Vancouver Sun, January 22, 2009, page A6.

?? is a partial recreation of Fig. 1 from [**Isenberg2017Vispubdata.orgMetadataCollection**], which is in the public domain.

Acknowledgments

The authors wish to thank WPI Data Science PhD student and visualization literacy researcher Karen Bonilla Janssens (advised by Professor Lane Harrison) for advice and guidance during the writing process.

template

A About Appendices

Refer to Sec. 9 for instructions regarding appendices.

B Troubleshooting

B.1 ifpdf error

If you receive compilation errors along the lines of `Package ifpdf Error: Name clash, \ifpdf is already defined` then please add a new line `\let\ifpdf\relax` right after the `\documentclass[journal]{vgtc}` call. Note that your error is due to packages you use that define `\ifpdf` which is obsolete (the result is that `\ifpdf` is defined twice); these packages should be changed to use `ifpdf` package instead.

B.2 `pdfendlink` error

Occasionally (for some LaTeX distributions) this hyper-linked bibTeX style may lead to **compilation errors** (`pdfendlink ended up in different nesting level ...`) if a reference entry is broken across two pages (due to a bug in `hyperref`). In this case, make sure you have the latest version of the `hyperref` package (i.e. update your LaTeX installation/packages) or, alternatively, revert back to `\bibliographystyle{abbrv-doi}` (at the expense of removing hyperlinks from the bibliography) and try `\bibliographystyle{abbrv-doi-hyperref}` again after some more editing.