Visualize Me: Explore the quickly updated world

Zining Wang
Northeastern University
first@wznmickey.com

***Abstract*—The process of scientific writing is often tangled up with the intricacies of typesetting, leading to frustration and wasted time for researchers. In this paper, we introduce Typst, a new typesetting system designed specifically for scientific writing. Typst untangles the typesetting process, allowing researchers to compose papers faster. In a series of experiments we demonstrate that Typst offers several advantages, including faster document creation, simplified syntax, and increased ease-of-use.**

I. Introduction and Background

Recent technological advancements have created a dynamic landscape where hardware and AI capabilities are evolving faster than ever. This growth introduces complexity in selecting the best-performing, cost-effective components for specific tasks. While benchmarking tools provide performance scores, they often overlook factors such as power consumption, price, and domain-specific requirements.

For instance, in hardware:

- Power consumption influences energy cost and thermal design.
- Price must be weighed against marginal performance gains.
- Use case specificity matters: single-core performance may benefit legacy applications, while multi-core performance is critical for parallel workloads.

Similarly, in the AI domain, users must consider inference speed, model size, and accuracy depending on their use case and resource constraints.

Existing benchmark aggregation platforms lack interactivity or customization options, making it hard to tailor insights to individual needs. “Visualize Me” aims to fill this gap.

“Visualize Me” represents a novel approach to bridging the gap between high-level performance data and user-specific needs, empowering individuals and organizations to make informed decisions when selecting components or AI models. It recognizes that performance scores alone don’t provide a holistic view of a system’s suitability for specific tasks.

Key features include:

- Data isn’t just presented in static charts. Users can interact with the data, zooming into specific components or comparing alternatives in real-time. This helps to understand the trade-offs between power consumption, price, and performance for hardware, or inference speed, accuracy, and model size for AI.
- Users can see how different choices might affect the system as a whole, instantly displaying the impact of adding more power, more cores, or different AI models to a specific workload.
- It’s not just about comparing hardware. You can also compare AI components, for example, a GPU against a TPU for a deep learning task, factoring in things like speed, accuracy, and model fit.

It can be used for various applications, including:

- Hardware Selection: A developer might need to pick a processor for a legacy system that benefits from strong single-core performance. They can filter through the options and immediately see which processors provide the best performance-to-cost ratio while considering power consumption.
- AI Model Deployment: A company looking to deploy an AI model on a resource-constrained edge device might use the platform to compare lightweight models’ inference speed, accuracy, and power usage to find the ideal trade-off.
- Data Center Optimization: A data center manager can leverage “Visualize Me” to pick the right processors, cooling solutions, and network setups that optimize both performance and energy consumption across multiple workloads.

By creating a dynamic, interactive, and customizable interface, “Visualize Me” transforms benchmarking from a static, one-size-fits-all process into an agile decision-making tool that aligns with specific performance, power, and cost requirements, making it ideal for both tech enthusiasts and professionals alike.

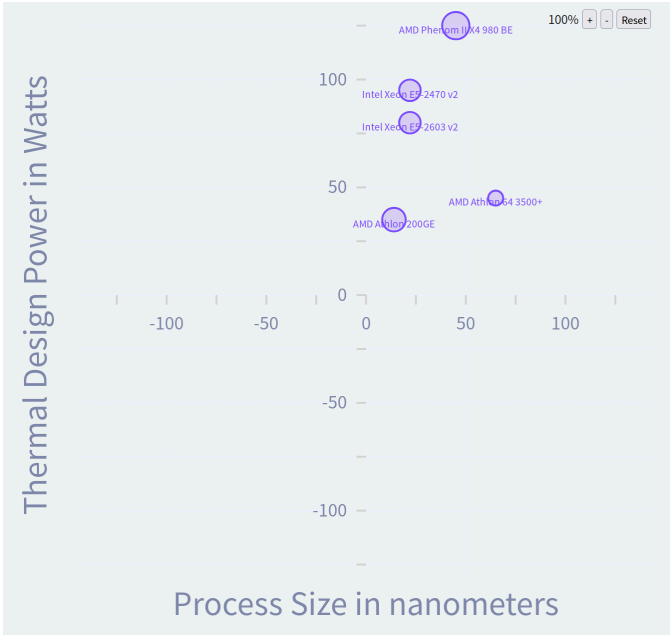

Fig. 1: Default view of the interactive scatter plot.

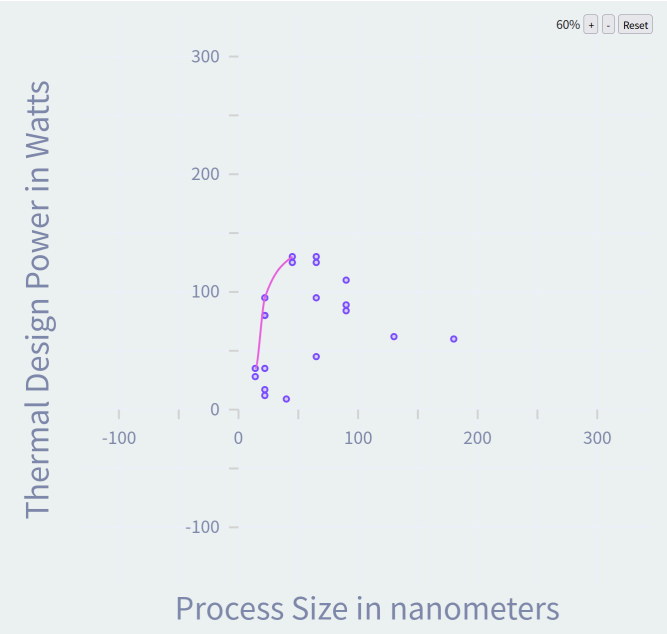

Fig. 2: Load CSV data and enable the Pareto line.

II. Features

The project is built in Elm, a functional programming language that compiles to JavaScript and HTML. Elm was chosen for its robust architecture, type safety, and ease of creating interactive SVG-based visualizations.

A. Zoom and pan capabilities

Users can zoom in and out of the data visualization, allowing them to focus on specific areas of interest. This feature is particularly useful for large datasets where users may want to examine details without losing the overall context.

B. CSV upload and remote data loading

Users can upload CSV files containing performance data or load data from remote sources. This flexibility allows users to work with their datasets without needing to modify the codebase.

C. Customizable axes and labels

Users can customize the axes and labels of the charts to suit their specific needs. This feature is essential for tailoring the visualization to different datasets or performance metrics.

D. Pareto front detection

The system can detect the Pareto front in the data, allowing users to identify optimal solutions based on multiple criteria. This feature is crucial for decision-making in scenarios where trade-offs exist between different performance metrics.

E. Toggleable size and text indicators

Users can toggle size and text indicators on the charts, providing additional context and information about the data points. This feature enhances the user experience by allowing users to choose how they want to visualize the data.

F. Responsive UI for dynamic data exploration

The user interface is designed to be responsive, allowing users to explore the data dynamically. This feature ensures that the visualization remains accessible and usable across different devices and screen sizes.

III. Implementation

The current system is divided into modules with core functionality implemented in Main.elm and Zoom.elm. Data is parsed from CSV files and visualized using SVG-based charts.

This document explains the implementation of an interactive scatter plot created using Elm. The plot allows users to zoom in and out, drag to pan, toggle data visibility, and visualize a Pareto line.

A. Model and Initial State

The Model type alias represents the application's state. It contains several fields to manage various aspects of the view:

- center: A point that represents the center of the zoomed area in the chart.
- dragging: This indicates whether the user is currently dragging the chart or not, represented by different states.
- percentage: Represents the zoom level (as a percentage).
- data: A list of points, each representing a data point on the chart. Each point has:
  - x: The x-coordinate.
  - y: The y-coordinate.
  - s: The size of the point.
  - w: A string associated with the point, such as a name or label.
- showSize, showText, showPareto: Boolean flags to toggle the visibility of point sizes, labels, and Pareto lines, respectively.
- textX, textY: Strings for the axis labels.
- url: A string to hold the URL from which the user can load CSV data.

The init function initializes the model with some default values, including example data points for CPUs with different specifications.

B. Messages

The Msg type defines various messages (or actions) that can trigger updates in the application state:

- Mouse Events: These messages handle mouse interactions such as mouse movement, mouse down, mouse up, and mouse leave, allowing the user to drag the chart around.
- Zooming: Messages like OnZoomIn, OnZoomOut, and OnZoomReset control zooming in, zooming out, or resetting the zoom level.
- File Operations: Messages like FileRequested, FileUpload, FileLoad are used for handling file uploads and loading CSV data from files.
- Toggling Visibility: The messages ToggleShowSize, ToggleShowText, and ToggleShowPareto toggle the vis-

ibility of point sizes, labels, and the Pareto line on the chart.
- Text Inputs: Messages such as UpdateTextX, UpdateTextY, and UpdateUrl handle the updating of input fields for the axis labels and URL.

C. Update Function

The update function processes each message and updates the model accordingly. Here's how the update function works for different messages:

- UpdateUrl: Updates the url field in the model with a new URL.
- LoadUrl: If the URL is not empty, it triggers the loading of CSV data from the provided URL using the Http.get request.
- GotResponse: Handles the response from the server when loading the file. If the response is successful, it decodes the CSV data and updates the model's data field.
- Mouse Events: For OnMouseDown, OnMouseMove, and OnMouseUp, the dragging logic is handled. The position of the center point of the chart is updated based on mouse movements, and the dragging state is updated accordingly.
- Zooming: For OnZoomIn, OnZoomOut, and OnZoomReset, the percentage field is adjusted to zoom in, zoom out, or reset the zoom level. The center point of the chart is also adjusted accordingly.
- File Upload: The FileRequested and FileUpload messages trigger a file upload, and the FileLoad message loads the CSV file content into the model.
- Toggle Visibility: The ToggleShowSize, ToggleShowText, and ToggleShowPareto messages toggle the visibility of size, text labels, and the Pareto line.
- Text Inputs: The UpdateTextX and UpdateTextY messages update the axis labels for the chart.

D. View Function

The view function renders the UI of the application based on the current model state. It generates HTML elements like checkboxes, buttons, input fields, and a chart to visualize the data.

- Checkboxes: There are checkboxes to toggle the visibility of point sizes (showSize), text labels (showText), and the Pareto line (showPareto).
- Input Fields: There are input fields for the user to enter the URL to load data from and for updating the axis labels.
- Zoom Controls: Buttons for zooming in, zooming out, and resetting the zoom level.
- Chart: The chart is rendered using the Chart library, with various configurations such as height, width, zoom percentage, and data series.
    - The chart is rendered as a scatter plot, where each point is represented by a circle. The size of each point can be toggled based on the showSize flag.
    - Labels for the x and y axes are displayed, and the showText flag controls whether the labels are shown on the data points.
    - The showPareto flag determines whether a Pareto line is shown on the chart, representing a cumulative line that helps visualize the Pareto principle.
- Zooming: The chooseZoom function listens for mouse wheel events (Wheel.Event), and depending on the scroll direction (deltaY), it adjusts the zoom level by increasing or decreasing the percentage value.

E. Pareto Logic

The getPareto function computes the Pareto line for the chart. It takes a list of points, sorts them by the x-coordinate, and then iteratively builds the Pareto front by checking if the current point has a higher y-value than the previous point in the list.

- Pareto Front: The Pareto front is a subset of the data points that are not dominated by other points. A point is considered to be "dominant" if no other point has both a higher x and a higher y value.
- Duplicate Removal: After building the Pareto front, the duplicates based on the x-coordinate are removed to ensure that each point in the Pareto front is unique in terms of its x-value.

F. Zooming Mechanism

The zooming mechanism works by changing the percentage value of the Model, which controls how much the chart is zoomed in or out. When the user scrolls the mouse wheel, the OnWheelEvent message is triggered, and the zoom level is adjusted accordingly.

- Zoom In: When the user scrolls up, the zoom percentage increases by 20%.
- Zoom Out: When the user scrolls down, the zoom percentage decreases by 20%, but it is capped to a minimum of 1% to prevent the chart from being zoomed out too much.
- Reset Zoom: A button allows the user to reset the zoom to the default value of 100%.

G. File Handling

The application supports loading CSV files, which is handled through the file upload mechanism (FileRequested, FileUpload, and FileLoad messages). The CSV data is decoded using the Csv.Decode library, and the decoded data points are added to the data field in the model.

H. Event Handling

The application is highly interactive, responding to events such as mouse clicks, movements, wheel scrolling, and input field changes. These events allow the user to manipulate the chart and customize its appearance.

I. Data Source

We preset the data from two source.

- https://www.kaggle.com/datasets/michaelbryantds/cpu-and-gpu-product-data
- https://llm-stats.com/ (manually get some as an example)

Users can also upload their own CSV files or load data from a URL by themselves. The CSV data is parsed and visualized in the chart.

J. Deployment

We open-source the code on GitHub, and users can run the application locally by cloning the repository and running the Elm application. The project is designed to be easily extensible, allowing developers to add new features or customize existing ones.

For quick using, we also convert the code to a html file and host it on GitHub Pages. Users can access the application directly from the browser without needing to set up a local environment by visiting the GitHub Pages link https://wznmickey.github.io/visualizeMe/

IV. Conclusion

"Visualize Me" empowers users to interactively explore complex performance data, improving decision-making for hardware buyers and AI researchers alike. The current implementation demonstrates the feasibility of a browser-based visualization tool using Elm. While limitations exist, the project establishes a solid foundation for further development and practical deployment.