# **VDR: Visual DOM Representation**

**Abstract**

This paper presents an innovative tool designed for beginner programmers to debug and understand HTML, CSS, and JavaScript code interactively. The tool integrates real-time syntax highlighting, code execution visualization, and AI-generated explanations for seamless learning. By bridging the gap between code and rendered output, the tool significantly enhances the debugging and learning experience. The case studies discussed in the paper demonstrate its effectiveness in educational settings, highlighting its strengths and limitations.

**Introduction**

Learning web development can be challenging for beginners, especially when understanding how HTML, CSS, and JavaScript work together to create interactive web pages. New programmers or developers often struggle with debugging their code and comprehending the flow of execution in event-driven programming. Traditional code editors and debuggers may not provide the intuitive visual feedback necessary for novices to grasp complex concepts effectively.

This paper introduces VDR (Visual DOM Representation), a tool specifically designed to aid beginner programmers in understanding and debugging web code interactively. VDR integrates real-time syntax highlighting, interactive code execution visualization, and AI-generated explanations to provide a seamless learning experience. By offering a visual representation of the Document Object Model (DOM) and highlighting the connections between code and output, VDR aims to bridge the gap between writing code and seeing its effects, enhancing beginners' learning and debugging process.

The following sections discuss the motivation behind developing VDR, explore related work in the field, deep dive into the workflow and components of the tool, and a brief case study explaining the tool's effectiveness amongst new developers and debuggers. Furthermore, the limitations of the current implementation are examined, concluding with insights into the impact of VDR on beginner programmers.

**Motivation**

The motivation for developing this tool is rooted in new developers' challenges in understanding web-based event-driven programming. Inspired by educational tools like Python Tutor, which aid beginners by visually breaking down code execution, this tool similarly seeks to simplify learning by focusing on web interactions. Moreover, the research into the tools discussed in the section below highlights the importance of visual feedback in learning programming. Studies emphasize that visualizations can significantly improve comprehension, particularly for first-year computer science students and new learners who want to grasp concepts in detail. By allowing users to view event flows from different perspectives, this approach also helps developers debug issues for single-page websites more easily by providing them with a detailed understanding of the flow of events.

**Related Work**

Several tools and research projects have contributed to the field of interactive and visual programming aids:

1. **Python Tutor**: Python Tutor allows users to visualize the execution of Python, JavaScript, and other programming languages step-by-step. It provides a detailed view of memory changes, including function calls and variable states, giving beginners a strong foundation in understanding code flow. However, Python Tutor focuses on line-by-line execution and does not offer a real-time view of event-based interactions in web development. (Guo, 2013).
2. **CrossCode**: CrossCode offers a multi-level visualization system for JavaScript, allowing users to move between different abstraction levels to understand the flow of code execution. (Hayatpur, Wigdor, & Xia, 2023). This tool is effective for visualizing control flow and data flow in JavaScript but lacks a specific focus on user-triggered events and DOM interactions that are crucial in web development. Its design aims at aiding developers in understanding complex code rather than real-time interaction flows.
3. **Anteater**: Anteater is designed for Python program execution and provides an interactive visualization system focused on debugging and understanding variable dependencies. Unlike the proposed tool, Anteater does not target web interactions or DOM structures, as its primary function is debugging through static visualization rather than real-time interaction visualization (Faust, Isaacs, et al., 2019).
4. **Method Execution Reports**: This tool generates summaries of method executions with textual and visual representations, helping developers interpret function behaviors (Beck, Siddiqui, Bergel, & Weiskopf, 2017). However, it does not support visualization of web events or interactive event-driven programming, which are critical for understanding user interactions in web development.
5. **InterCode**: InterCode standardizes and benchmarks interactive coding environments, enabling consistent evaluation of coding tools that provide execution feedback (Yang, Prabhakar, Narasimhan, & Yao, 2023). While this framework is helpful for comparing coding tools, it lacks a focus on visualizing specific event flows and DOM interactions that are central to web development.
6. **Systematic Review on Software Visualization**: A literature review on modern software visualization methods highlights the broad approaches used in visualizing software structures, behavior, and evolution. The review categorizes visualization tools but indicates a gap in tools specifically designed for real-time web development environments, where users need instant feedback on event-driven actions within the DOM. (Chotisarn, Merino, Zheng, et al., 2020)

**Gaps in the research**

Despite advancements in these tools, there remains a gap in providing a unified platform that integrates real-time visualization of event-driven web programming, DOM interactions, line highlights and AI-generated explanations tailored for beginners. VDR aims to fill this gap by offering an interactive environment where users can see the immediate effects of their code on the web page, understand event flows, and receive explanations to enhance their learning experience.

**Workflow**

VDR is designed to provide an interactive coding environment where beginners can write, visualize, and understand HTML, CSS, and JavaScript code. The tool comprises several components that work together to enhance the learning and debugging experience.

Highlighting tags Code Injected

Contacting OpenAI and injecting Explanation

JavaScript functions Code injected

HTML Line number Injected

Normal Code entered by the user

Enhanced Code base

**Steps to Run the tool**

Download the file form GitHub link <https://github.com/goki-1/VDR/blob/main/index.html>

And add the OpenAI API keys in the code at line number 440 and save it and double click the file. The VDR tool will open in your default web browser. Now write or upload the Html, CSS, JavaScript files and press “Run Code” button. You will see two outputs, go to Enhanced output and interact with the website to see difference

**Components**

1. **User Interface (UI)**The UI is split into two main sections: the editor section and the output section. The editor section allows users to write or upload their HTML, CSS, and JavaScript code using CodeMirror, a versatile text editor implemented in JavaScript for the browser. The output section displays the rendered web page in two tabs: the normal output and the enhanced output.
   * **Editor Section**: Contains tabs for HTML, CSS, and JavaScript editors, each providing syntax highlighting and line numbering to help users keep track of their code.
   * **File Input**: Users can upload code files directly into the editor, facilitating the use of existing code snippets.
   * **Run Button**: Executes the code and updates the output section with the latest changes, allowing users to see immediate results.
   * **Enhanced Output Section**: Displays the enhanced version of the rendered web page. In this tab, users can interact with the web page elements to see which lines of code correspond to which parts of the page. When an element is clicked, the tool highlights the relevant code in the editor and displays AI-generated explanations, helping users understand the relationship between their code and the output.
2. **HTML Line Injector for Highlighting Lines**When the user runs their code, VDR processes the HTML content to add data-line-number attributes to HTML elements. This is done by splitting the HTML code into lines and injecting a data-line-number attribute into each opening tag. This attribute corresponds to the line number in the editor and is essential for linking the visual elements on the web page back to the code.

  
  
**Component Highlighter in the Enhanced Website**In the enhanced output, the tool adds event listeners to all HTML elements with a data-line-number attribute. When a user clicks on an element in the rendered web page, the corresponding line in the HTML editor is highlighted. Additionally, the element itself is temporarily highlighted to provide visual feedback, helping users connect the visual output with the underlying code.

A screen shot of a computer

Description automatically generated  
  
**JavaScript Function Injector for Highlighting Lines**The tool processes the JavaScript code to inject calls to a displayJsLineNumber function at the beginning of each executable line. This is done using **Babel**, a JavaScript compiler. The injected function sends a message to the parent window to highlight the corresponding line in the JavaScript editor when that line is executed, allowing users to follow the execution flow, by injecting function calls in each function and class.  
  
 function addDisplayJsLineNumber(jsCode) {

const instrumentedCode = Babel.transform(jsCode, {

plugins: [/\* Custom Babel plugin to inject displayJsLineNumber calls \*/]

}).code;

return `

function displayJsLineNumber(lineNumber) {

window.parent.postMessage({ language: 'javascript', lines: [lineNumber] }, '\*');

}

${instrumentedCode}

`;

}

1. **Structured Output API Calls to OpenAI**To provide AI-generated explanations, the tool sends the user's code to the OpenAI API using a specially crafted prompt. This prompt is designed to get detailed explanations for each line of code in a structured format that the tool can easily use.Detailed Explanation of the Prompt.The prompt consists of two messages: a system message and a user message.

* **System Message**: This sets the context for the AI assistant. It tells the AI to act as a top professor who provides explanations for HTML, CSS, and JavaScript code. It specifies that the AI should return the explanations as a JSON object, where each key is the line number of the HTML tag (matching the data-line-number), and the value is the explanation related to that line. The output should look like {line:5, explanation:"Explanation here"}.
* **User Message**: This includes the actual code that needs to be explained. It tells the AI that the user is a beginner in programming and requests clear explanations based on the provided HTML, CSS, and JavaScript code. This structured approach makes it easier for the tool to display explanations when a user interacts with the rendered web page. When the AI returns the explanations as the array of objects where each line number corresponds to html tag, the tool parses the JSON object and integrates it into the enhanced output. This way, when a user clicks on an element on the page, the tool can immediately display the relevant explanation.

A screenshot of a computer program

Description automatically generated

1. **Integration with CodeMirror Libraries**The explanations and highlights are synchronized with the CodeMirror editors. When a user interacts with the rendered web page, the tool communicates with the editors to highlight the relevant lines of code, providing an immediate connection between the code and its visual output. This integration ensures a cohesive user experience where code editing, and output visualization are tightly linked.

**Case Studies**

**Comparison with W3Schools**

W3Schools is a popular resource for learning web development. It provides tutorials, references, and examples for HTML, CSS, and JavaScript. However, it lacks interactive debugging tools that link code directly to the rendered output.

* **W3Schools Approach:**
  + Provides code examples and "Try It Yourself" editors.
  + Users can modify code and see the output, but there is no linkage between the code editor and specific elements in the output.
  + Does not offer AI-generated explanations or highlight corresponding code when interacting with the output.
* **VDR Approach:**
  + Offers an integrated environment where code editors and output are connected.
  + Enhanced output allows users to click on elements and see exactly which lines of code are responsible.
  + VDR highlights the tag clicked with red border so that it is clear which tag is responsible for that particular CSS effects
  + Provides AI-generated explanations for each element and associated code, aiding in understanding and debugging.

In the case study three different code bases comprising of Html, CSS and JavaScript code were tested, while W3Schools might help the student understand the syntax and provide similar examples, it doesn't assist in pointing out line numbers and components on website as user clicks anywhere on the website. VDR helps the student identify the problem by linking the non-functioning button directly to the code and explanations, facilitating a quicker and deeper understanding.

**Example Case Study**

When tested on simple Shop website where users can add items into card and see the total in the cart and click on checkout and empty the card, our tool made it easy to understand different tags, associated CSS and JavaScript associated with it.

A screenshot of a shopping cart

Description automatically generated

.When clicked on Checkout button

A computer code with text

Description automatically generated

. Html tag is highlighted

A screen shot of a computer program

Description automatically generated

. Along with JavaScript function that this button triggered

A black background with white text

Description automatically generated

. Also, we get explanation regarding the button which explains it functionality, CCS and JavaScript code associated with it

On the other hand, in websites like W3 school’s user only get online compiler and no other assistance

**Debugging Case Study**

VDR also helps in detecting bugs faster in the websites. When tested on Portfolio website which has CSS issues, we were able to pinpoint the problem.

1. **Loading the Code into VDR**The student loads their HTML, CSS, and JavaScript code into VDR's respective editors and clicks the "Run" button.
2. **Observing the Enhanced Output**In the enhanced output tab, they interact with the button. VDR provides immediate visual feedback by highlighting the lines of code associated with the button or normal tag when clicked Along with highlighting the tag with red border on the website.
3. **Receiving AI-Generated Explanations**When the student clicks on the button in the enhanced output, an explanation panel appears, showing the AI-generated explanation for the button element and its associated JavaScript code.
4. **Identifying the Issue**The student notices that the AI explanation mentions the button with ID and the event listener. They also observe that the line numbers in the explanations correspond to their code.

In the images below heading “Projects” is not aligned at the centre of the webpage, when clicked on it, it highlighted the whole tag, with corresponding line in the codebase. Here the culprit is heading tag which should have additional CSS properties aligning as column

**A screenshot of a computer

Description automatically generated** after click**A screenshot of a project

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

A screenshot of a project

Description automatically generatedFix: Adding into .projects CSS

*flex-direction: column;*

*align-items: center;*

After making changes and running it again  
**Difference Between GPT-4 and GPT-4 Mini Outputs**

In VDR, AI-generated explanations are obtained by sending code to the OpenAI API using the GPT-4 or GPT-4 Mini models.

* **GPT-4:**
  + Offers more detailed and accurate explanations.
  + Better at understanding context and providing beginner-friendly language.
  + Can handle more complex code snippets effectively.
* **GPT-4 Mini:**
  + Provides quicker responses due to reduced computational requirements.
  + Explanations might be less detailed or nuanced compared to GPT-4.
  + May struggle with complex code, resulting in less accurate or helpful explanations.

**Example Comparison:**

* *Using GPT-4:*
  + **Line 6 (HTML):** "This line creates a button element with the ID 'colorButton'. It's intended to be interactive and will trigger a function when clicked."
  + **Associated JavaScript:** "Adds an event listener to the button. When clicked, it changes the body's background color to light blue."
* *Using GPT-4 Mini:*
  + **Line 6 (HTML):** "Creates a button with ID 'colorButton'."
  + **Associated JavaScript:** "When the button is clicked, changes background color."

The GPT-4 model provides more context and explanation, which can be more helpful for beginners trying to understand the intricacies of their code. GPT-4 Mini, while faster, may not offer the same depth of understanding.

**Limitations**

While VDR offers several benefits for beginner programmers, it has certain limitations:

1. **Cannot Upload Multiple Files**The current implementation allows users to upload only one file per code type (HTML, CSS, or JavaScript). This limitation means that projects with multiple HTML pages or external scripts cannot be fully represented within the tool, restricting its use to simpler projects.
2. **Single Page Website**VDR is designed to work with single-page web applications. It does not support multi-page websites or applications that require navigation between different pages, limiting its applicability for more complex projects that involve routing and page management.
3. **Does Not Work for Larger Codebases**The tool is not optimized for large codebases. Processing large amounts of code, especially when generating AI explanations, can lead to performance issues or exceed API limits, making it less suitable for complex applications. This limitation affects the tool's scalability for advanced users.
4. **Code Should Be Well Formatted**The code provided by the user needs to be well-formatted and correctly indented. Since the tool relies on line numbers and code parsing, poorly formatted code may lead to incorrect highlighting or parsing errors, reducing the effectiveness of the tool. Users must ensure their code adheres to standard formatting practices.

**Future developments**

Future developments could focus on addressing the limitations by enabling support for multiple files, optimizing performance for larger codebases, and improving code parsing robustness. VDR has the potential to become an essential tool in programming education, fostering a deeper understanding of web development among beginners.

**Conclusion**

This paper introduced VDR (Visual DOM Representation), a tool designed to aid beginner programmers in understanding and debugging HTML, CSS, and JavaScript code interactively. By integrating real-time syntax highlighting, code execution visualization, and AI-generated explanations, VDR bridges the gap between code and rendered output, enhancing the learning experience.

The motivation behind the tool highlighted the challenges faced by new developers in web-based event-driven programming. A review of related work identified a gap in existing tools that VDR aims to fill. The workflow section detailed how each component of VDR contributes to its overall functionality, providing users with immediate visual feedback and explanations.

While VDR has limitations, such as not supporting multiple files or larger codebases, it serves as a valuable educational resource for beginners. By offering an interactive environment where users can see the immediate effects of their code, understand event flows, and receive explanations, it helps users grasp complex concepts more effectively.

**References**

Chotisarn, N., Merino, L., Zheng, X., & et al. (2020). A systematic literature review of modern software visualization. *Journal of Visualization, 23*(4), 539–558. <https://doi.org/10.1007/s12650-020-00647-w>

Faust, R., Isaacs, K., Bernstein, W. Z., Sharp, M., & Scheidegger, C. (2019). Anteater: Interactive visualization of program execution values in context. *arXiv preprint*. <https://arxiv.org/abs/1907.02872>

Hayatpur, D., Wigdor, D., & Xia, H. (2023). CrossCode: Multi-level visualization of program execution. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Article 593, pp. 1–13). Association for Computing Machinery. <https://doi.org/10.1145/3544548.3581390>

Yang, J., Prabhakar, A., Narasimhan, K., & Yao, S. (2023). InterCode: Standardizing and benchmarking interactive coding with execution feedback. *arXiv preprint*. <https://arxiv.org/abs/2306.14898>

Beck, F., Siddiqui, H. A., Bergel, A., & Weiskopf, D. (2017). Method execution reports: Generating text and visualization to describe program behavior. In *2017 IEEE Working Conference on Software Visualization (VISSOFT)* (pp. 1–10). <https://doi.org/10.1109/VISSOFT.2017.11>

Hijón-Neira, R., Pizarro, C., French, J., Paredes-Barragán, P., & Duignan, M. (2023). Improving CS1 programming learning with visual execution environments. *Information, 14*(10), 579. <https://doi.org/10.3390/info14100579>

Guo, P. J. (2013). Online Python Tutor: Embeddable web-based program visualization for CS education. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE ’13)*. Association for Computing Machinery.