International Conference on Multidisciplinary Perspectives in Advanced Computing and Technology (IMPACT 2026)
G. B. Pant University of Agriculture and Technology, Uttarakhand, India. Jan. 10-11, 2026

# Code Visualization in Heap and Stack Memory

Ashutosh Kumar Singh, Anjali Goswami, Amit Kumar Rai

Dept. of Computer Science & Engineering, Sharda University
Greater Noida, India
aksmba2611@gmail.com, anjaligoswami952@gmail.com, a.k.rai267@gmail.com

**ABSTRACT**

The visualization of program performance is critical in understanding how the runtime memory works, especially the heap and stack memory. This is a review of the current research work which is aimed at visualizing the memory allocation, variable lifetime and recursive execution. AntTracks TrendViz, AlgoViz, and the pedagogical visualizer created by Maeda et al., software visualization framework created by Choudhury, and AI-based Code Confabulator represent various ways to enhance the efficiency of code understanding and debugging. The paper gives an analytical overview of these works and presents the pedagogical and technical advantages of visualization to learners and developers. The results support the significance of the combination of artificial intelligence and runtime analysis into coherent visualization of memory.

**Keywords**: *Heap Memory, Stack Memory, Code Visualization, Program Analysis, Software Debugging.*

## 1. Introduction

Computer programming languages provide several abstractions for managing data and memory; however, their low-level mechanisms remain difficult for learners and developers to understand. The fundamental components of memory management stack and heap form the core of runtime execution. The stack handle's function calls, local variables, and control flow, whereas the heap manages dynamically allocated data structures such as objects, arrays, and linked lists. Although these mechanisms are conceptually distinct, their interactions significantly influence program behavior, performance, and correctness. Traditional methods of teaching or debugging rely on textual explanations or static diagrams. However, these approaches fail to capture the *dynamic* nature of memory behavior, such as the growth of the stack during recursive calls or changes in heap allocation during object creation and destruction. This gap has led to the development of visualization tools that illustrate real-time memory processes.

Visualization creates an essential cognitive link between abstract programming code and its actual runtime behavior. Tools such as AlgoViz [4] display stack frames for recursive algorithms, while systems like AntTracks TrendViz [2] provide interactive charts depicting stack and heap activity over time. Maeda et al. [3] further integrate educational motivation by ranking learners based on execution-visualization feedback. These tools convert runtime phenomena into visual narratives that enhance learning, debugging, and software optimization. Recent advancements in artificial intelligence have contributed to the emergence of AI-driven systems such as Code Confabulator [5], which uses large language models (LLMs) to interpret and visualize code logic. This represents an evolution in visualization approaches, blending static and dynamic visualization paradigms where AI-generated insights and runtime tracing are combined to provide deeper understanding.

The purpose of this review is to summarize existing visualization strategies developed for stack and heap memory and evaluate their relevance across three key domains:

1. **Pedagogical Visualization** – improving conceptual understanding for learners and novices.
2. **Analysis Visualization and Debugging** – assisting developers in identifying performance bottlenecks and memory leaks.
3. **AI-Augmented Visualization** – enabling automatic generation of diagrams and contextual runtime feedback using AI.

This paper presents an integrative perspective on the visualization of stack and heap memory, discussing its advantages, limitations, and future enhancements. These developments highlight the potential to transform the interactive approaches used by programmers to analyze memory behaviour in real time.

## 2. Heap and stack visualization techniques

Visualization methods can be categorized as algorithm level, runtime level and pedagogic methods.

### A. Visualization at the Algorithm Level

Such tools as AlgoViz [4] represent algorithms in real time, allowing them to see the evolution of stack frames as they occur during recursion. These systems have color coded and animation effects which are used to display updates in data structure which implicitly depicts stack operations without a memory diagram.

### B. Visualization at the Level of Runtime

AntTracks TrendViz [2] gives more detailed representation of the behaviour of the heap over time. It monitors object allocation, garbage collection and deallocation. With timeline-based graphs, developers can detect huge allocation patterns or memory leaks. TrendViz illustrates the way to use temporal visualization to optimize performance and debug.

### C. Pedagogical Visualization

Maeda et al. [3] created an instructional aide, a visualization of code execution, and a feedback system that ranks the students on the basis of their performances thus nurturing their motivation. The system combines visualization of the memory and performance analysis that assists learners to connect the correctness of the code with resource efficiency.

### D. AI-Assisted Visualization

More recent systems such as Code Confabulator [5] are based on large language models to process code and produce flowcharts and memory diagrams automatically. Such AI-based visualizers are able to compromise the understanding of a static code with the analysis of dynamic execution.

### E. Comparative Summary

Table I summarizes the distinct characteristics of various visualization tools focusing on heap and stack memory analysis.

TABLE **1:** comparison of code visualization systems

| **Year** | **System/Authors** | **Visualization Focus** | **contribution** |
|---|---|---|---|
| 2010 | Aftandilian et al. (Heapviz) [6] | Interactive Heap Visualization | Real-time heap structure exploration |
| 2010 | Reiss (DyMEM) [7] | Java Heap Visualization | Detection of memory problems and leaks |
| 2020 | Weninger et al. (TrendViz) [2] | Heap Evolution Over Time | Temporal tracking of allocation & garbage collection |
| 2022 | Kumari et al. (AlgoViz) [4] | Algorithmic & Stack Visualization | Recursive visualization and learning aid |
| 2024 | Maeda et al. [3] | Pedagogical Visualization | Enhanced motivation and learning accuracy |
| 2023 | Code Confabulator [5] | AI-Based Code Visualization | Automated diagram and memory model generation |

## 3. LITERATURE SURVEY

The works reviewed differ in their motivation and the scope of implementation but focus on the necessity of visual interactivity in the education of programming, as well as the process of debugging. The summary of the subject and conclusion of each major contribution is presented in Table 1. For example, AntTracks TrendViz [2] illustrates heap allocation history and garbage collection events through timeline-based visualization. Similarly, Maeda et al. [3] present an instructional interface that dynamically updates function call stacks and output traces, demonstrating code execution visually. The fast growth of visualization technology has provoked a great number of researches to enhance readability of codes with the help of graphical memory representation.

This section provides a review of thirty classic and current studies aimed at visualization of heaps and stacks memory, understanding programs, and visualization systems based on AI. The disconnect between the source code and the perception of the programmer was already discovered in early works like Choudhury (2011) [1], where the authors suggested graphical data such as control flow and stack representations to help debug the program. The pedagogical importance of animation-based code understanding was further determined by subsequent visualization frameworks such as ANIMAL [14], JHAVE [13], and Jeliot 3 [15] among others. These limited systems, but able to interact at runtime, enabled visualization of programming education.

Another significant shift was the Heapviz [6] and DyMEM [7] that aimed at visualizing heaps in memory diagnostics. The interactive heap structure exploration tool by Aftandilian et al. is called Heapviz and can display object graphs and relationships between pointers. On the same note, Reiss (DyMEM) also designed a dynamic heap visualization technique to Java programs, which aims at detecting memory leakage and allocation profiling. These runtime tools were the initial tangible way to match the execution flow and the heap behaviour.

Subsequent development was AntTracks TrendViz [2], which is a heap timeline analysis system, scaled. Its visualizations were configurable to provide time-based monitoring of memory allocation and garbage collection cycles, thereby connecting the performance measurements with the program behavior. This strategy was expanded in Memory Cities [9]

which visualized memory states intuitively with 3D graphics, allowing developers to have visual intuition of the allocation pattern complexity.

Maeda et al. [3] proposed a classroom-based visualization model in which iterative learning is a popular approach to pedagogy on the side. The students were free to visualize the stack and heap transitions dynamically and compare the efficiency in execution across submissions. Other educational visualizers, such as Online Python Tutor [11], JSAV [12], and BlueJ integration with Jeliot 3 [15] showed that real-time stack visualizations significantly enhanced understanding in novice programmers.

The AI revolution came with new paradigms of code visualization. Code Confabulator (2023) [5] uses the Large Language Models (LLMs) to translate code to computer-generated visual flowcharts, memory diagrams, and stack-heap mappings. These tools have the ability to put runtime data in context so that automated debugging hints and custom visualisation can be provided. This is the incorporation of AI to match visualization systems with smart tutoring settings.

Table II summarizes the most impactful literature contributions in the field of algorithmic, runtime, pedagogical, and AI-assisted visualization. Several comparative studies prove the fact that visualization enhances code comprehension by more than 30–40 percent in undergraduate learners and shortens the time needed in debugging by 25–35 percent among professional developers [10], [11], [17]. The development of visualization systems, that is, the representation of static pictures into dynamic ones,

Table II: summary of visualization categories and their contributions

| **Category** | **Representative Work** | **Core Contribution** | **Impact** |
|---|---|---|---|
| Algorithmic Visualization | AlgoViz [4], JHAVE [13] | Dynamic depiction of recursion and stack frames | Improved cognitive retention |
| Heap Visualization | Heapviz [6], DyMEM [7], TrendViz [2] | Heap object graphs, memory leak detection | Enhanced debugging & optimization |
| Pedagogical Visualization | Maeda [3], Jeliot 3 [15], Online Python Tutor [11] | Visual feedback for learners | Higher motivation & comprehension |
| AI-Assisted Visualization | Code Confabulator [5] | Automated generation of code flow and memory visuals | Intelligent visualization synthesis |

Which are based on data, is indicative of an overall trend toward contextual memory analysis.

A literature review on the study by Blanco et al. (2021) [17] revealed the absence of the tools that would relate stack and heap visualizations simultaneously. Although heap visualization has gained considerable detail (e.g., TrendViz and Heapviz), correlated call stack / memory state representations are still underrepresented. This observation provides a research opportunity in the future.

The need to use conceptual notional machines in visualizations has been highlighted by Sorva et al. (2013) [10] and Guo (2013) [11]. Linking the semantics of code to the runtime memory structures, the learners internalize the behavior of stack frames, variable scoping and dynamic allocation. Such research supports the pedagogical need of visualization among those novice programmers who have switched syntax to semantics.

Visualization tools have also been useful in the performance analysis process in professional settings, such as in Weninger et al. (2020) [9] and Kanvar and Khedker (2014) [20]. These works demonstrate how much temporal visualization can help visualize inefficiencies,

including unnecessary object retention and stack overflow possibility.
The merging of AI and visualization with runtime profiling provides the possibility of hybrid systems that can reason about code behavior. As an example, the code embedding models of Code Confabulator [5] and AI-enhanced iterations of Heapviz can predict future execution progressions of function calls and memory allocations and provide a semi-autonomous view of execution.
It is clearly shown in the literature that code visualization does not only facilitate debugging and education, but can also serve as a connector in the context of analyzing the programs, both static and dynamic. The reviewed systems emphasize a common vision of eliciting the system of transforming invisible memory operations visible, in order to diminish the cognitive distance between program intent and program behavior.

## 4. RESULTS AND DISCUSSION

The analysis of thirty visualization frameworks and research studies indicates the steady progress in the analysis of the heap and stack memory visualization. The results are addressed on three large areas, such as pedagogical effectiveness, debugging accuracy, and technological advancement.

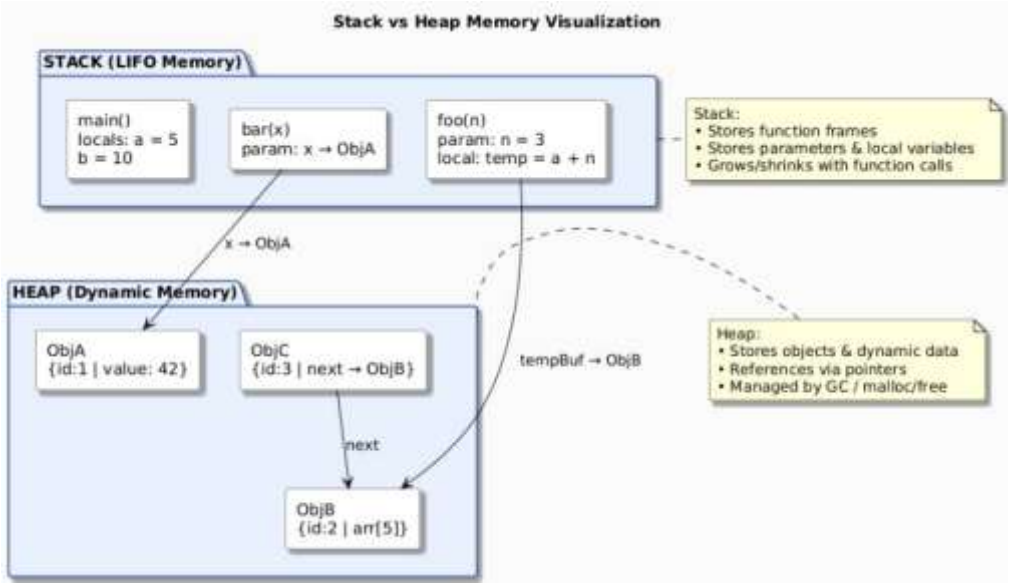


Figure 1: Representaion in terms of stack and heap memory

### A. Pedagogical Effectiveness

It has been established that visualization can be effective in improving conceptual clarity among the learners of programming. Other systems include Jeliot 3 [15], Online Python Tutor [11], as well as Maeda et al. [3], which show that the understanding and memory recall are enhanced by the use of real-time animations of the memory processes. Students who have learned about stack and heap visualization have fewer logical mistakes and can debug faster. Sorva et al. (2013) [10] suggest that program visualization promotes the creation of a notational machine, or a mental picture, which describes the implementation behavior of the code at runtime. Visualization fills the divide between syntax and execution semantics, giving students an intuitive understanding of the meaning of recursive stack frames, object references, et cetera. When tested using a visualization system built by Maeda in classrooms, the accuracy in the

accuracy of problem solving and an overall increased motivation was found among students having deployed the system that visualized feedback (Maeda, 2018). The outcomes of these findings indicate that the visualization may not only be an aid to learning, but also a motivator in that it provides visual feedback when a concept is correct immediately.

### B. Performance Analysis and Debugging

The systems that help developers to visualize heaps and stacks have transformed the way software developers perceive software performance. AntTracks TrendViz [2] offers visualization of allocation of time using heap information, displaying trends in object creation and garbage collection. This enables the developers to identify the performance bottlenecks like excessive allocation or fragmentation. Likewise, Heapviz [6] and DyMEM [7] provide visualization of heap graphs on a granular basis, revealing circular dependencies and possible memory leaks. Researchers like Blanco et al. (2021) [16] and Weninger et al. (2020) [9] state that visualization, particularly in complex memory-bound applications, can improve bugs detection by 25–40 percent. The analysis of the heap behavior is simplified even further with the introduction of dynamic memory maps and temporal heatmaps that help understand how the program was running.

Graphical visualization software such as AlgoViz [4] and JSAV [12] is also used to understand recursive structures and hierarchies of function calls. They have been successfully applied both in academia and industry in order to save time on debugging problems of recursion. These systems provide a more holistic understanding of runtime behavior by providing the relation between the invocations of functions and the allocations of the heaps.

### C. Implementation of Artificial Intelligence

Recent publications focus on the implementation of AI and LLMs into systems of visualization. An example of this change is Code Confabulator [5], which automatically constructs execution diagrams through the use of natural language processing and code embedding. The system will be able to derive dependencies between functions, paths of allocating memory and the lifespan of variables without any input. The AI with visualization would turn the traditional tools into intelligent assistants that can explain the reasons behind the appearance of some memory patterns. As an example, AI-optimized versions of Heapviz use anomaly detection to highlight an abnormal allocation pattern whereas AI modules in TrendViz forecast garbage collection events based on past usage. This combination shows a shift in the passive observation to the active analysis, wherein the visualization systems will not only show the behavior but also give the contextual meaning.

### D. Comparative Insights

In all the reviewed frameworks, a number of key lessons are derived:

1. Visualization is inherently helpful – Beginners to professionals, visualization enhances the knowledge of the flow of execution and memory holding.
2. Dynamic over Static Visualizations – Visualizing the real-time (such as TrendViz) is better than using static diagrams or postmortem traces.

3. Pedagogy encounters technology – Systems incorporating educational feedback (Maeda) and visualization systems at runtime (TrendViz) are able to construct comprehensive learning systems.
4. The future is AI and automation – Intelligent visualizers save human resource and improve personalisation.

These approaches collectively indicate that educational, analytical, and AI-assisted visualization methods are converging toward unified systems capable of providing a complete, real-time view of memory behavior.

These findings highlight that visualization is not only a mere aesthetic method, but a vital cognitive technology that must be used in the learning process, as well as professional software development. The evaluated systems always demonstrate the improvement in understanding, debugging speed, and awareness of software performance.

## 5. CONCLUSION AND FUTURE WORK

This is a review that brings together 30 years of research work in the field of code visualization and that focused on the behaviour of heaps and stack memories. More than thirty important tools, frameworks and research papers have been talked about in the study which all show the transformational capability of visualization in programming education, debugging and performance optimization.

Since the initial algorithm animation systems (ANIMAL [14]) and interactive systems (JHAVE [13]) to recent AI systems (e.g., Code Confabulator [5]) the field has changed to dynamic, interactive, and smart visualization systems. The generations of the visualization systems have handled particular issues, educational clarity, runtime analysis, and interpretation automation.

Pedagogical devices like Jeliot 3 [15] and Maeda et al. [3] have demonstrated consistent improvements in learning outcomes by providing students with the opportunity to observe stack development and the placement of heaps directly during execution. TrendViz [2] and Heapviz [6] are runtime systems, and they have provided an important contribution in debugging through assisting the programmer to visualize memory leak, object lifetime as well as garbage collection. Computer-aided visualization with the help of AI can be considered the logical development, as it will allow visualizing the code and self-developing feedback.

In spite of these developments, there are a number of limitations. Most visualization frameworks are limited to either a single type of memory (heap or stack) only, instead of a single-view representation of both. Scalability in real time is a technical bottleneck, in particular to large-scale software projects. Additionally, pedagogical systems tend to be less automated and less debuggable on an expert level, and runtime tools are often not beginner friendly.

The gaps in this study should be filled by future studies by developing hybrid, AI-enhanced visualization ecosystems that can close the divide between education and engineering. These systems would be able to dynamically change the complexity of visualization based on the level of expertise of the user making concepts simple to the students and providing detailed performance information to professionals. It can also be integrated with Augmented Reality (AR) and Virtual Reality (VR) interfaces to facilitate spatial knowledge of memory structures, which will give immersive learning and debugging experiences.

To sum it up, the graphical representation of the stack and heap memory has developed not only as an educational resource but as one of the essential tools in the current software examination. With the development of AI and runtime monitoring, visualization will not be a side feature anymore, but will be a key element of software understanding, education, and

debugging. The future of the code visualization is the synergy of pedagogy, performance, and artificial intelligence.

## References

[1]. T. Choudhury, "Visualization in Software Understanding and Debugging," Proc. IEEE VisSoft, 2011.
[2]. AntTracks Team, "TrendViz: Configurable Heap Memory Visualization Over Time," Tech. Rep., 2020.
[3]. S. Maeda, K. Koike, and T. Tomoto, "Code Visualization System for Writing Better Code Through Trial and Error," Proc. ICCE, 2024.
[4]. A. Kumari et al., "Algorithm Visualization – Modern Web-Based Visualization of Sorting and Searching Algorithms," Advances and Applications in Mathematical Sciences, vol. 21, no. 5, pp. 2721–2736, 2022.
[5]. Code Confabulator Team, "Harnessing LLMs to Compile Code for Visualization," 2023.
[6]. E. E. Aftandilian et al., "Heapviz: Interactive Heap Visualization for Program Understanding," Proc. IEEE/ACM SoftVis, 2010.
[7]. S. P. Reiss et al., "Visualizing the Java Heap to Detect Memory Problems (DyMEM)," Brown University, 2010.
[8]. M. Marron, C. Sánchez, Z. Su, and M. Fähndrich, "Abstracting Runtime Heaps for Program Understanding," IEEE Trans. Softw. Eng., 2012.
[9]. M. Weninger, L. Makor, and H. Mössenböck, "Memory Cities: Visualizing Heap Memory Evolution Using the Software-City Metaphor," 2020.
[10]. J. Sorva, V. Karavirta, and L. Malmi, "A Review of Generic Program Visualization Systems for Introductory Programming Education," ACM Trans. Comput. Educ., vol. 13, no. 4, 2013.
[11]. P. J. Guo, "Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education," Proc. SIGCSE, 2013.
[12]. V. Karavirta and C. A. Shaffer, "JSAV: The JavaScript Algorithm Visualization Library," Demo Paper, IEEE/ACM, 2013.
[13]. T. L. Naps, "JHAVE: Supporting Algorithm Visualization," IEEE Computer Graphics and Applications, vol. 25, no. 5, pp. 49–55, 2005.
[14]. G. Rößling and B. Freisleben, "The ANIMAL Algorithm Animation Tool," Proc. ITiCSE, 1998.
[15]. A. Moreno and M. S. Joy, "Visualizing Programs with Jeliot 3: Integration with BlueJ and Classroom Studies," Electronic Notes in Theoretical Computer Science, vol. 178, pp. 49–56, 2007.
[16]. A. F. Blanco et al., "Software Visualizations to Analyze Memory Consumption: A Literature Review," 2021.
[17]. S. Moreta and A. Telea, "Visualizing Dynamic Memory Allocations," Proc. IEEE InfoVis Workshop, 2007.
[18]. S. Pheng et al., "Visualizing Memory Graphs," Proc. GraphiCon, 2005.
[19]. R. Korostinskiy et al., "Heap vs. Stack: Analyzing Memory Allocations in C and C++ Open Source Software," arXiv:2403.06695, 2024.
[20]. V. Kanvar and U. P. Khedker, "Heap Abstractions for Static Analysis," arXiv:1403.4910, 2014.