

Thinking beyond chatbots’ threat to education: Visualizations to elucidate the writing and coding process

Badri Adhikari

adhikarib@ums1.edu

Department of Computer Science, University of Missouri-St. Louis, USA

Abstract

The landscape of educational practices for teaching and learning languages has been predominantly centered around outcome-driven approaches. The recent accessibility of large language models has thoroughly disrupted these approaches. As we transform our language teaching and learning practices to account for this disruption, it is important to note that language learning plays a pivotal role in developing human intelligence. Writing and computer programming are two essential skills integral to our education systems. What and how we write shapes our thinking and sets us on the path of self-directed learning. While most educators understand that ‘process’ and ‘product’ are both important and inseparable, in most educational settings, providing constructive feedback on a learner’s formative process is challenging. For instance, it is straightforward in computer programming to assess whether a learner-submitted code runs. However, evaluating the learner’s creative process and providing meaningful feedback on the process can be challenging. To address this long-standing issue in education (and learning), this work presents a new set of visualization tools to summarize the inherent and taught capabilities of a learner’s writing or programming process. These interactive *Process Visualizations* (PVs) provide insightful, empowering, and personalized process-oriented feedback to the learners. The toolbox is ready to be tested by educators and learners and is publicly available at www.processfeedback.org. Focusing on providing feedback on a learner’s process—from self, peers, and educators—will facilitate learners’ ability to acquire higher-order skills such as self-directed learning and metacognition.

Keywords: Process Visualizations, process-oriented learning, self-directed learning, metacognition

1 Introduction

1.1 Metacognitive writing and coding

Writing. Writing shapes our thinking, regardless of whether we are writing to learn or learning to write [1, 2]. During the process, learners write in order to clarify their thinking [3]. Writing facilitates a logical and linear presentation of ideas that allows writers to explain their points of view. Emig [4], a prominent 20th-century English composition scholar, was best known for her foundational work relating to the process theory of composition, where writing is presented as a process rather than a product. Her work provided a fundamental way to develop new ideas by teaching students to develop their writing skills [4]. The effective teaching of writing is an essential component of any successful school program. Writing skills are best taught when related to specific and relevant content, which requires a more analytical approach to writing and provides a particularly welcoming context for thinking deeply. However, the recent use of online resources and chatbots reduces the motivation to write for users who let the chatbot write for them. The role of writing in the development of logical thinking is as important now as it has ever been.

Coding. Most schools and universities worldwide require students to learn computer programming. Several studies have pointed out that learning to code can improve various skills. In 1984, Pea and Kurland [5] identified a difference in learning programming compared with other studies. Their objective was to improve the *pedagogy of programming*. They were among the first to teach computer programming, as they worked with high school students trying to determine the best approach to enabling students to see beyond the programming, so they could eventually acquire analogical, conditional, procedural, and temporal reasoning skills, as well as higher cognitive skills. They were looking for a *cognitive style* that identified a student as being successful or not successful in performing specific programming subtasks. However, they never labeled

any of the learners they analyzed as unable to learn to program. They emphasized teachers being trained to respond appropriately to a student’s cognitive learning style. In a subsequent work, however, the idea was refuted [6]. A recent review revisiting the topic finds that learning to code can improve a learner’s skills in mathematics, problem-solving, critical thinking, socialization, self-management, and academic strategizing [7]. These findings suggest that the influence of learning to write a computer program on a learner’s cognitive abilities can be anywhere from trivial to profound. It could simply be learning a vocabulary of commands and rules for arranging the commands. Or, it could be far more than programming, and acquiring powerfully general higher cognitive skills such as planning abilities. Irrespective of the magnitude of impact, learning to build useful software can serve as a means for learners to attain and improve widely applicable cognitive skills.

Learning metacognition. Metacognition is commonly defined as the ability to think about one’s own thinking. The text which inspired this definition was provided by John Flavell in 1979. He introduced it as the learners ‘knowledge of their own cognition’ and formally defined metacognition as the ‘knowledge and cognition about cognitive phenomena’ [8, 9]. Using metacognitive thinking and strategies enables learners to become flexible, creative, and self-directed. Metacognition assists learners with additional educational needs in understanding learning tasks, self-organizing, and regulating their own learning. The National Research Council’s book, titled *How People Learn: Brain, Mind, Experience, and School* [10], stated:

“Metacognition also includes self-regulation—the ability to orchestrate one’s learning: to plan, monitor success, and correct errors when appropriate—all necessary for effective intentional learning... Metacognition also refers to the ability to reflect on one’s own performance.” (National Research Council, 2000, p. 97).

The most intriguing and difficult part of teaching metacognition is that thinking is idiosyncratic. Brame (2019) [11], however, tells us not to be afraid to ask questions and to work on our own self-directed, self-monitoring, and self-evaluating plans. Effective self-directed planning involves understanding the overall goal and timeline of a project at hand. Similarly, self-monitoring involves investigating why one failed to master a needed skill, and self-evaluating involves assessing why one was or was not successful and efficient. Overall, learners can develop higher-order skills when they ask themselves the questions needed to help them “plan, monitor, and evaluate” their learning experiences.

Metacognitive writing and coding. Writing and coding can now be outsourced to generative deep learning systems. In the coming age, when we are increasingly outsourcing repetitive cognitive tasks to deep learning systems, it is crucial for learners to develop higher-order cognitive skills. To learn to solve problems in authentic situations, learners should develop metacognitive abilities to manage and regulate their problem-solving process. For example, in the domain of mathematics, metacognition is a fundamental component of the problem-solving process [12]. Ironically, these cognitive tasks, such as writing and coding, are learners’ vehicles to organize their thoughts and develop several higher-order thinking skills. Metacognition does not develop on its own for all learners; thus, it must be a target of learning supported explicitly in the classroom [13]. Langer (1986) [14] found that students at all grade levels were deficient in higher-order thinking skills. Today’s outcome-oriented learning and education priorities make it even more challenging to develop such skills. In general, developing metacognition and other related skills such as self-awareness, understanding our own mind, and critical thinking, require practicing reflection, mindfulness, learning from experience, engaging in self-experimentation, and seeking feedback. A potential solution to improving metacognition in learners is process-focused thinking aimed at developing process-oriented metacognition.

1.2 Process-oriented feedback

Importance of process-focus. Research on learning, directly or indirectly, embraces the “process pedagogy.” For instance, in direct opposition to the focus on written products, the 1970s and 1980s observed a groundswell of support for “process” approaches to teaching writing [1]. In writing, learners and educators use words such as drafting, prewriting, and revising in commonplace speech [3]. Similarly, in learning computer programming, educators often require learners to submit process data such as code comments,

challenges faced, and code optimizations as a part of the assignment requirements. Some programming approaches, such as ‘test-first’ encourage programmers to write functional tests before the corresponding implementation code can be entered [15]. If process-oriented teaching and learning are so essential, why are today’s education systems mostly outcome-oriented? One key reason is that assessing the process and providing feedback is not straightforward.

Importance of feedback. Feedback is one of the most powerful influences on learning and achievement [16]. If effective, feedback can promote student investment and independence in writing and coding [3]. Detailed and descriptive feedback is found to be effective when given alone, unaccompanied by grades or praise. And surprisingly, the perceived source of feedback (a computer or an instructor) is found to have little impact on the results [17]. Peer feedback can also be effective. For example, a study to examine the effects of peer-assessment skill training on learners’ writing performance found that learners who receive in-depth peer assessment outperform those who do not [18]. Similarly, self-assessment can have a powerful impact on learner motivation and achievement [19]. Self-assessment is much more than simply checking answers. It is a process in which learners monitor and evaluate the nature of their thinking to identify strategies that improve understanding [20]. Within all the kinds of feedback that can be provided to learners, feedback on the process can be considerably more powerful.

Importance of feedback on the process. Providing feedback on a learner’s thought process can have a more positive impact on learning than feedback focused on the final outcome [21]. Across a wide range of educational settings, formative-type assessments and feedback do ‘work’ and effectively promote student learning [22]. They provide valuable information to both learners and educators [19]. Formative assessments, in general, are a tenet of good teaching. Unfortunately, approaches to assessment have remained inappropriately focused on testing [23]. Pressures on education are threatening the use of formative assessments [24]. Teachers are, and continue to be under enormous pressure to get their learners to achieve [25]. Moreover, in practice, these formative assessments make learners follow a prescribed and explicit process with *a lot of steps*, potentially developing *learned dependence* [26]. In addition, the practice of *lock stepping* learners by forcing them to divide their effort into stages conflicts with the recursive nature of the creative endeavor processes [3]. However, a more significant challenge is that providing feedback on the process instead of the product is difficult.

1.3 Computational technologies to cultivate process feedback

Lack of computational technologies to provide feedback on the process. Learners can be provided with process feedback in several areas of the creative process. Process activities in writing are often subdivided into stages such as prewriting, drafting, revising, and editing. These processes are recursive rather than linear and complex rather than simple [1]. To help learners *prepare to write* or *prepare to code*, educators can offer learners models of finished writing or code, or more effectively, can provide models on the spot, in front of learners, for a task in question. Models that demonstrate *how* to plan and write are more effective than the ones that demonstrate *what* to write [3]. In other words, in addition to providing professional examples, locally produced examples and examples written by other learners of similar age and skill level help learners imagine how to craft a product incorporating similar features [3]. However, effective tools to explore, analyze, and learn from the process that leads to the product, are missing. The crux of the problem lies in the absence of effective computational technologies for today’s educators to demonstrate and discuss their own process of generating a product or to present and analyze a *model* process and for today’s learners to acquire feedback on their process.

Overview of this work. This article presents, discusses, and demonstrates how maintaining a self-monitored revision history (*snapshots*) of one’s own text as a writer or a programmer’s code at frequent intervals during the writing or coding process, can be used to build interactive data visualizations that display and summarize the process. These Process Visualizations can provide insights into the steps a writer or a programmer took during their writing or coding process. This ongoing self-review by the author can facilitate effective self-feedback, peer feedback, and educator feedback.

2 Methodology and Implementation

This section describes the overall approach of retaining a user’s process data (typed text of a writer or code or a programmer) and using that data to render informative data visualizations. First, the learner must realize that the web application being used was designed to allow users to write text or code. Once a user starts to type, the web application stores the text/code typed every five seconds in the user’s local storage (Internet browser storage). These timestamped data contain the revision history of the user-typed text over the entire typing period. In other words, every entry recorded in this revision history contains both a timestamp and the text the user was working with during time stamp process. Together, the full revision data collection tool captures how the typed text/code evolves (or changes) over time.

Connecting passages over versions. Transforming typed text data (e.g., an English essay) with revision history information (i.e., time and full text at that time) into a format that can be used to develop visualizations requires that each text passage at any given time point can be connected to the text in the following time point. In this case, a passage is defined as either a sentence or a paragraph. These passage-to-passage connections between two versions of the text can then be used to observe the writer’s creative flow when writing a passage, i.e., how a passage evolves (or changes) over time. For instance, as a writer adds content to a paragraph, the content of that paragraph will change with each time point. However, the various versions of a paragraph must be identified as the same to correctly display the paragraph’s origin (i.e., the time point when it was introduced). An algorithm that usefully maps passages in a text at time t with passages in the next step (at time $t + 1$) should also account for the fact that new passages can be added anywhere in the middle of the text at any time during the writing process.

Maintaining passage identities. Identifying the same passages over multiple text revisions is a crucial step toward generating meaningful visualizations. The idea is to initially assign a random identity (ID) to each passage in each text revision, i.e., all passages are assigned different IDs regardless of their similarity. Next, these passage IDs are updated by calculating their similarities. Specifically, the update process starts from the most recent pair of two text versions. First, similarities between all passages at the time point $t + 1$ and all those at time t are calculated (i.e., pairwise matched). For the two passages p and q , that have the highest similarity score, where p and q are passages at time $t + 1$ and t respectively, the ID of passage p is updated to that of q if the similarity is greater than a predefined similarity threshold. In other words, passage IDs are propagated from the most recent time point to previous time points. This process is repeated for all adjacent time point pairs all the way to the first pair of revisions (see **Algorithm 1**).

Passage-to-passage similarity using n -grams. The similarity between a pair of passages is calculated using the normalized dot product of the n -grams of the two passages. Such a purely statistical similarity calculation based on n -grams is language independent and works well when the lengths of the two passages are arbitrary [27]. Character-level n -grams are obtained with the n set to 5. Mathematically, if p and q are two passages being compared, where p_i is an n -gram in p and q_i is an n -gram in q , the first step is to calculate the weight of each distinct n -gram. In a passage m (either p or q), if m_i is the number of occurrences of a distinct n -gram i among J distinct n -grams, the weight of i can be calculated as,

$$x_i = \frac{m_i}{\sum_{j=1}^J m_j}, \quad (1)$$

where

$$\sum_{j=1}^J x_j = 1. \quad (2)$$

Then, for the two passages p and q , if x_{pj} is the relative frequency with which the n -gram j occurs in passage p and x_{qj} is the relative frequency with which the n -gram j occurs in passage q (out of all possible J n -grams in p and q), the normalized dot product can be calculated as,

$$SIMILARITY(p, q) = \frac{\sum_{j=1}^J x_{pj}x_{qj}}{\sqrt{\sum_{j=1}^J x_{pj}^2 \sum_{j=1}^J x_{qj}^2}}, \quad (3)$$

Algorithm 1 Identify same passages (paragraphs or sentences) between two versions of text

Inputs:

S , list of text data at various time points; each item consists of an ordered sequence of passages with IDs
 $threshold$, similarity threshold (between 0 and 1)

Outputs:

S' , IDs of passages at time t_n replaced with matching passages at time t_{n+1}

```
1: procedure UPDATEIDS
2:   for each pair of timestamps  $t$  and  $t + 1$ , starting from the latest pair do
3:     for each  $m \in S_t$  do ▷  $m$  is a passage at time point  $t$ 
4:        $n_{max} \leftarrow -1$  ▷ Stores the similarity of the most similar passage in  $t + 1$ 
5:        $n_{id} \leftarrow null$  ▷ Stores the ID of the corresponding passage
6:       for each  $n \in S_{t+1}$  do ▷  $n$  is a passage at time point  $t + 1$ 
7:         if SIMILARITY( $m, n$ ) >  $n_{max}$  then
8:            $n_{max} \leftarrow$  SIMILARITY( $m, n$ )
9:            $n_{id} \leftarrow n$ 's ID
10:        end if
11:      end for
12:      if  $n_{max} > threshold$  then
13:         $m_{id} \leftarrow n_{id}$  ▷ Update  $m$ 's ID so it is now identified as the same passage
14:      end if
15:    end for
16:  end for
17: end procedure
```

Preprocessing code data. Transforming the revision history data of code data (e.g., a Python program), into a format that can be used to develop visualizations has a different set of challenges. Unlike natural language text, code lines can have high character similarities and yet have considerably different meanings. For instance, in code, the only difference between an inner for loop line and an outer for loop line serving different purposes can be two characters, i and j . Because of this small coding difference, performing an all-to-all line comparison between two versions of code is ineffective. Consequently, the approach that was used to split natural language text paragraphs into sentences cannot be applied to split a code into lines. Instead, given two versions of the unsplit code, a ‘difference’ calculating algorithm is applied to find common subsequences or shortest edits [28] on the entire code blocks. The central idea in calculating this ‘difference’ is to formulate this problem of finding the longest common subsequence (LCS) and shortest edit script (SES) (known as the LCS/SES problem) as an instance of a single-source shortest path problem. The ‘npm diff’ library implementation is used to obtain these code segments each with a specific label, ‘common,’ ‘added,’ or ‘removed.’ These differences are obtained by considering each line as a unit, instead of considering each character or word as a unit. Following the idea similar to the one applied to natural language text versions, to obtain connections between the lines of code into adjacent versions of code, we initially generate random identities for all lines in all versions of the codes and update the identities of the sentences traversing backward from the most recent version of the code.

Parameter-driven (re-)rendering of visualizations for analysis. Interactive visualizations are rendered based on user-supplied values for parameters. This includes parameters such as n -gram size for calculating passage-to-passage similarity and the similarity threshold for determining the identities of passages (or lines in the case of code). Since all natural languages may not have space as the word separating character, *user-defined* word delimiting character(s) and sentence delimiting character(s) make the tool accessible across a wide variety of languages.

Technologies used in the web application. The single-page React web application was developed using several recent technologies. Tailwind and JavaScript were used for building the website. The site includes CKEditor for allowing users to type text, and Judge0 API [29], an open-source online code execution system, for allowing users to code, Monaco text editor and diff viewer for typing and displaying code, and Plotly for displaying visualizations. The web application is hosted in Cloudflare. A user’s process data can either be downloaded and stored for the user’s record only or saved to Cloudflare’s R2 buckets. This data is encrypted using the Advanced Encryption Standard (AES) implemented in the ‘npm crypto-js’ library, with

a user-supplied passcode, and can only be accessed with the passcode to decrypt it. This effectively serves as authentication for accessing the data. Data saved on the cloud R2 buckets are additionally encrypted using Cloudflare’s encryption-at-rest (EAR) and encryption-in-transit (EIT) technologies.

3 Results

The revision history data collected from a user can be cleaned, processed, and summarized to generate informative summary tables and interactive visualizations. Descriptive statistics, such as the total number of typed characters, words, sentences, lines, paragraphs, total and active time spent on typing, and average typing speed, can be presented as summary sentences or a table (see Figure 1). These quantitative summaries can be accompanied by data graphics displaying the actual data points (as shown in Figure 7). Similarly, the visualizations of the process data can facilitate drilling into the revisions; they also provide an in-depth analysis of the overall process followed by a writer or a programmer. The next section discusses several interactive and non-interactive Process Visualizations (PVs).

Descriptive statistics	
Characters:	2022
Words:	358
Sentences:	27
Paragraphs:	4
Total Time Spent:	29 minutes
Total Active Typing Time Spent:	16 minutes (More than 30sec of no typing is considered as inactive)
Average Typing Speed:	34 words/minute, 182 characters/minute (During active typing time)
Timestamps:	149

Figure 1: An example table of **descriptive statistics**.

PV1: Process playback (Figure 2). A highly accessible and easy-to-understand technique of visualizing a learner’s process is to play back their typing. A short (e.g., 30 seconds) playback can display the typing steps that a writer or a programmer took. Options to pause, stop, and replay at a user-controlled speed can make such a visualization interactive. During playback, at each time point, trails of text deleted (if any) and the trails of text added can be highlighted so a viewer can track where the learner was typing at the time point. Such a process playback visualization may not provide any quantitative information but can be an engaging initial step toward further analysis.

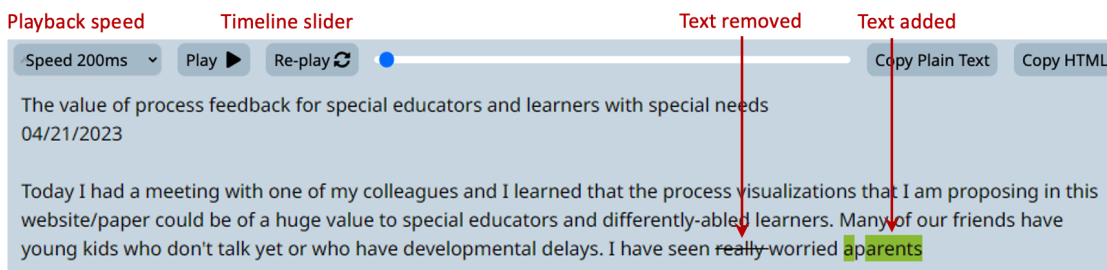


Figure 2: An example interface for **playing back** a writer’s writing process. The green-highlighted copy shows text added at the time point and strikethrough formatting corresponds to text removed. The slider allows a user to control and view at any pace.

PV2: Paragraph/sentence/line changes over time (Figure 3). When analyzing one’s writing or a programming process, it can be of interest to learn when a particular passage, paragraph, or sentence, in the case of writing and line in the case of code, originated or was removed. For instance, a sentence written at the beginning of the first paragraph may be pushed down throughout most of the writing time, only to later expand as a full paragraph. To visualize such process details at a passage level of interest, *stacked area plots*

may be used with unique colors assigned to each passage [30]. Such an area plot can also show the relative size of the sentences or paragraphs and how they change over time. Additionally, these stacked area plots can be interactive, i.e., a viewer can see the actual text of the passage at any given time point by hovering over the stacks.

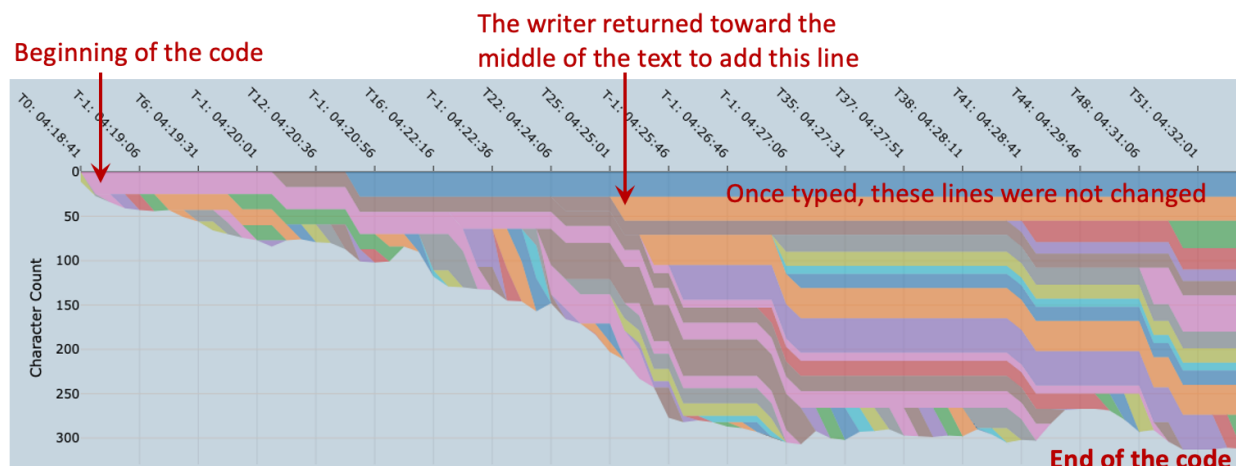


Figure 3: An example stacked area plot displaying **sentence changes over time**. This graph displays at what time point all the sentences originated.

PV3: Highlight active paragraph/sentence/line at each time point (Figure 4). An interactive stacked bar diagram can display each passage as a stack with time marked by x-axis. Such a detailed visualization can pinpoint (highlight) which passage a learner was working on at a specific time point during the writing or coding process. This visualization can also show how often a learner went back and revised a previously typed text. This can be achieved by highlighting only the passage being edited at that time point. An example application occurs when a learner is working on a revision task (starting with a bulk of text). In this case, the learner is looking at a stacked bar diagram to see if some paragraphs remain untouched.

PV4: Word frequencies (Figure 5). In writing analysis, observing the frequencies of words typed by a learner can be informative. For instance, self-directed learners may be interested in observing if they overuse the article, *the*, the adverbs *very* and *respectively* or conjunctive adverbs like *however* and *consequently*. Such observations may motivate them to increase their vocabulary and check the accuracy and necessity of overused and misused words. Although word frequency data can be calculated from the final text independent of the process, the frequency of words removed can also be informative. One caveat of counting words only based on user-defined character(s) but without the knowledge of a language is that these counts can be inaccurate for languages such as English where the same letters can be in lower or upper case. For instance, in such language-independent word frequency analysis, “the” and “The” would be identified as separate words. Word frequency can be plotted as standard bar diagrams displaying a certain percentage or a certain number of most used words and their corresponding usage frequency throughout the document.

PV5: Passage-to-passage pairwise similarities (Figure 6). Writers can also fall into the habit of repeatedly using the same phrases that fail to inform or can be deleted, such as *it should be noted that* or *it is well known that*. Visualizing pairwise similarity between all sentences in the final version of the text can help identify such habits. A visualization that serves this purpose is an interactive heatmap where each cell represents a similarity score between pairs of sentences. Displaying the actual sentence pairs by allowing them to hover over the cells can add interactivity.

PV6: Words/characters change over time (Figure 7). Changes in the number of words or characters over time can be visualized as a line diagram for cumulative and as a lollipop chart (or a bubble chart) for non-cumulative review. In a bubble chart, the bubble sizes can represent the number of characters added or removed. One approach to display bubbles at a consistent location in the y-axis is to set the y-axis based on the size of the change. Since changes can have a large variation data can be log-normalized.

PV7: Interactive any-to-any revision version comparator (Figure 8). Users interested in an in-depth



Figure 4: Example stacked bar diagrams **highlighting active paragraph(s) (top figure) and active sentence(s) (bottom figure) at each time point**. Each color corresponds to a paragraph in the top figure and a sentence in the bottom figure. Idle times are ignored in the timeline. Hovering over any of the stacks displays the actual paragraph or sentence at the time point. Highlighted stacks represent the user's paragraph and sentence changes at the corresponding time point.

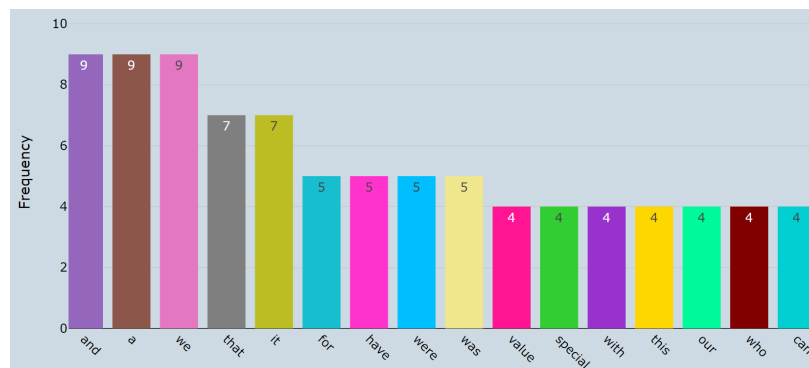


Figure 5: An example bar diagram displaying **word frequencies**.

analysis will want to observe the text added or removed between any two time points in the revision history. Users also need an interactive visualization tool that enables them to browse the entire timeline and select a

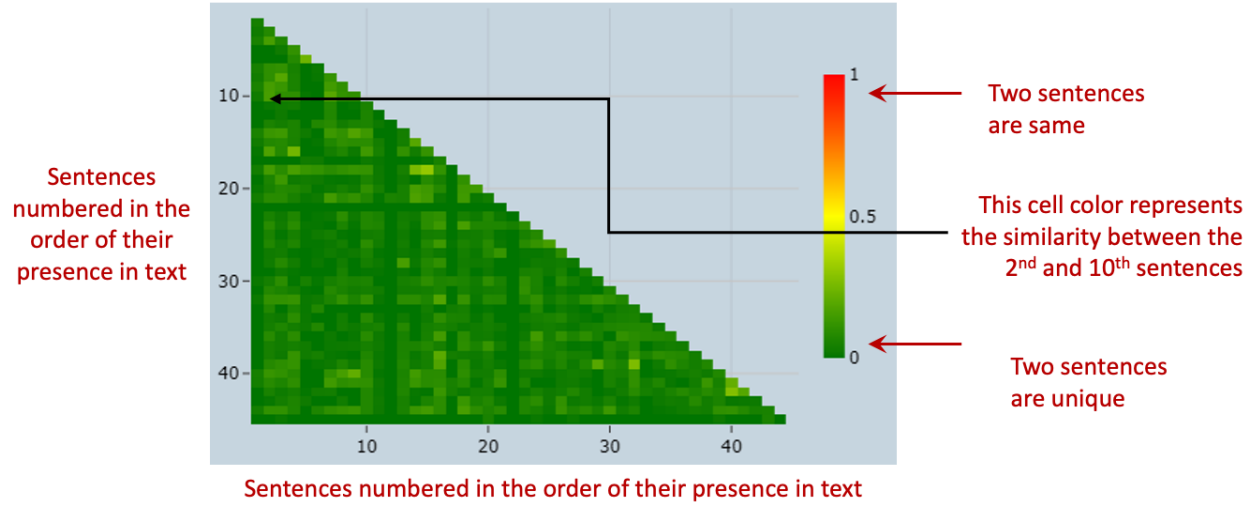


Figure 6: An example heatmap plot for displaying **pairwise passage-to-passage similarity**. Both x-axis and y-axis represent the sentences shown in the order in which they appear in the final writing. Hovering over any cell of the heatmap shows the two sentences and their similarity score.

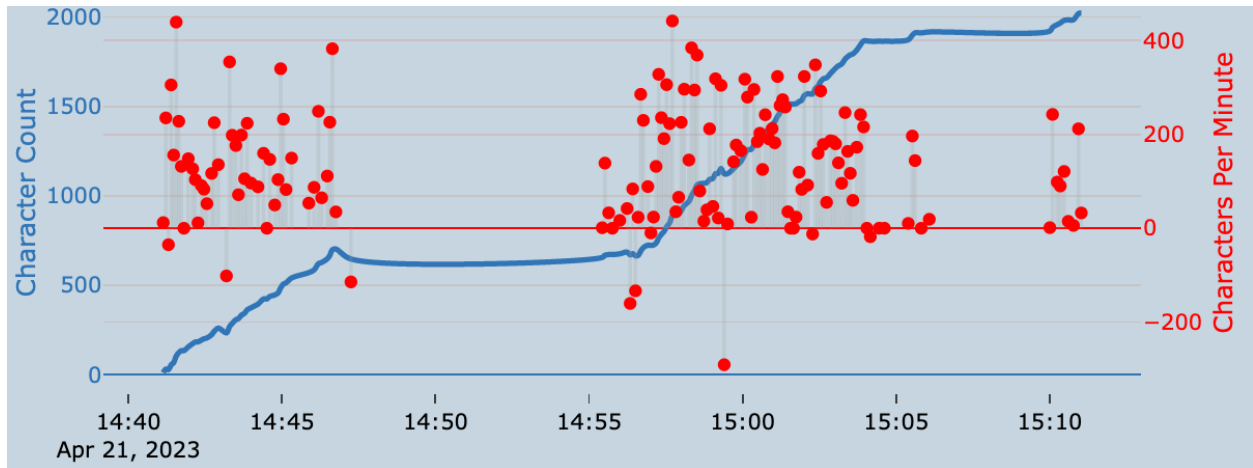


Figure 7: An example dual-axis scatter and line plot displaying an increase in the **total characters typed** over time points (blue line) and **characters per minute** at any time point (red points).

range for viewing the changes. To do so involves effectively selecting two time points. This can be achieved by combining three tools: 1) a difference viewer (i.e., the Monaco diff viewer), 2) a timeline chart (i.e., a bubble chart) which can show all the time points, and 3) a date-based navigator. Users can then interact through the bubble chart or navigate directly through the diff viewer. Such an interactive visualization can facilitate in-depth analysis, allowing the user to ask why a sentence or phrase was removed at a certain time.

PV8: Timeline showing successful and unsuccessful code executions (Figure 9). In the case of programming tasks, displaying a timeline plot with points indicating the executions of code, highlighting both successful and unsuccessful executions, can help users review how often they executed their code.

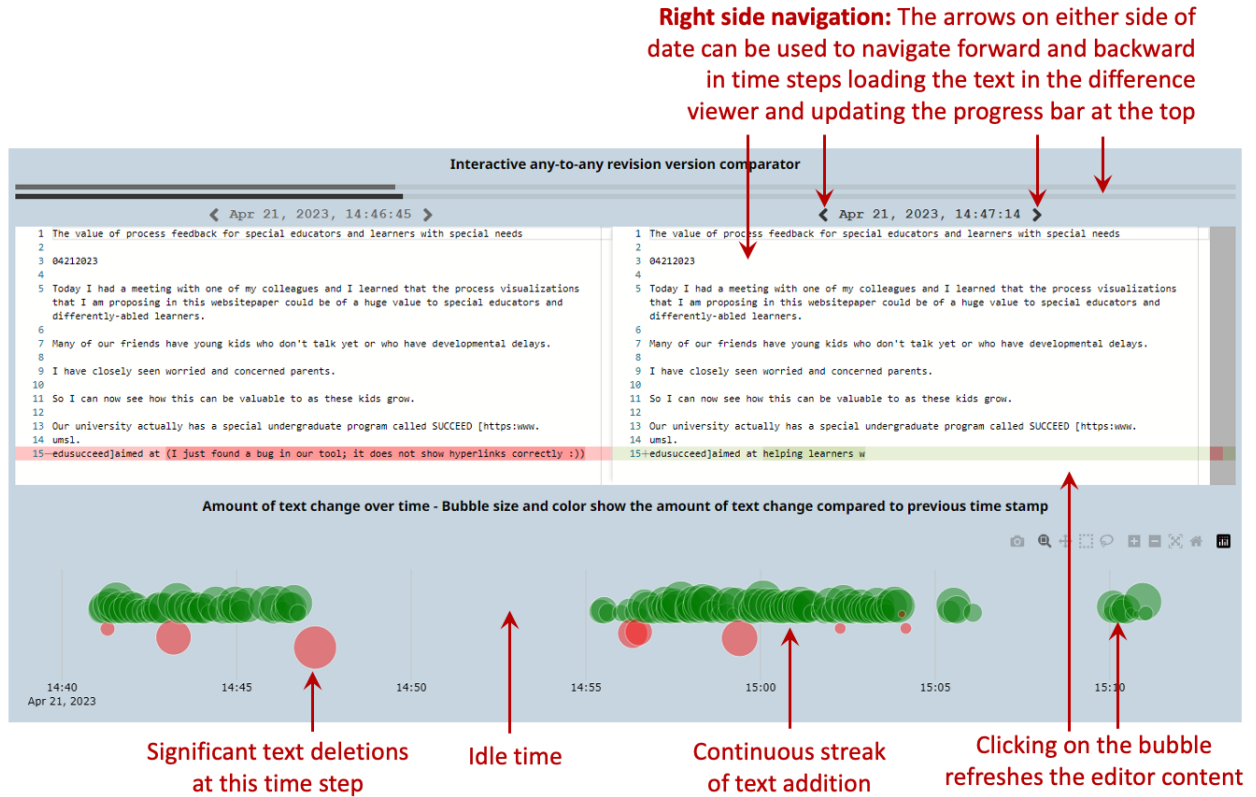


Figure 8: An example **interactive any-to-any revision version comparator** consisting of three components: 1) two date navigators at the top with arrows on either side of the dates, with progress bars on top connected to the dates (top progress bar for left date and bottom progress bar for right date); 2) a difference viewer shows two versions of text dictated by either the clicking on the date arrows or clicking on the bubble chart points at the figure's bottom; 3) a bubble chart shows the timeline where the bubble sizes correspond to the number of characters added or removed. Panning and zooming on the bubble chart facilitates easy micro/macro analysis. Clicking on any bubble loads the changes made in that particular version in the difference viewer windows. Green bubbles indicate that the version has text added and red indicates text was deleted. Also, hovering over the bubbles shows total characters added/removed.

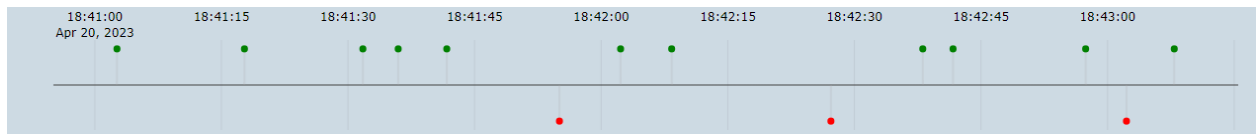


Figure 9: An example timeline showing **successful executions** (green points) and **executions with errors** (red points) in a computer programmer's coding process.

4 Discussion

4.1 Exploratory data visualizations as clarifying tools

Data graphics are widely used as an explanation tool. In a more general, all-inclusive context, data graphics can serve the purpose of explanation, exploration, or both. While explanatory data graphics can communicate a point or display a pattern or a concept, exploratory data visualizations invite the viewer to discover

information [31]. While discussing the fundamental principles of analytical design, in his book “Beautiful Evidence” [32], Edward Tufte lists “completely integrating words, numbers, images, and diagrams” as one of the core principles. Although these exploratory data visualizations can appear complex at first, they can be vehicles for new insights. Alberto Cairo in *Truthful Art* [33], noted insightfulness as one of the most important qualities in his presentation of *five qualities* of great visualizations. Adding interactivity to visualizations can strengthen this integration. Interactive and integrative visualizations can facilitate two fundamental acts of science: description and comparison. Several aspects of interactive data visualization can, however, make them inaccessible. For increased public accessibility, these visualizations should be improved for user-friendliness and high usability.

4.2 Acquiring feedback from insightful Process Visualizations

The Process Visualizations clarify the concepts and steps of the process. This provides tools for acquiring feedback—from self, a peer, or an educator—on several dimensions of a writer’s or programmer’s process. For instance, if a writer is focused on improving their prewriting skills, these visualizations can enable exploring how frequently the writer switches to other modes, such as revising and editing (see Figure 4). While it may be recommended to finish prewriting and then focus on revision, it is possible to polish paragraphs while writing. Prewriting embodies the initial steps of writing. A writer’s approach is usually an iterative process; thus, it can be insightful to learn what process a writer follows to finalize an initial prewrite. Similarly, in learning and practicing the ‘test first’ approach to programming, Process Visualizations can enable exploration of how often the programmer revises the test cases as they start to code the actual solution and how often they revise the solution as the test cases evolve. In general, learners could also use these visualizations for self-revelation, direction, and improvement, and the visualizations can provide immediate answers to questions such as: (1) where in the process did they spend most of the time, (2) how much time did they spend on creating the first draft vs. revising and editing the draft, (3) which paragraphs were edited and revised, and (4) which paragraphs were not edited. Overall, the visualizations facilitate exploring, looking back, and looking into one’s process and approach. Furthermore, learners can engage in *inquiry* by focusing on the process, regardless of whether the processing was originated by the learner, a peer, or an educator. Exploring and understanding the theories behind teaching and learning followed by the engagement of all parties toward making the most of every learning experience is the portal to learning through communication, feedback, improvement, and success. Engaging in inquiry and agreeing on the *theories that drive teaching and learning* are powerful. Whitney [3] summarized theorizing and engagement by saying:

“Engaging in inquiry means not only learning practices recommended by others or perfecting the practical execution of a set of teaching strategies but, rather, theorizing about teaching and learning in a way that then frames future interpretation and decision-making.” (Beyond strategies: Teacher practice, writing process, and the influence of inquiry, 2008, p. 205).

4.3 Securing personal process data

Individual creations such as a piece of writing, code, or art are intellectual properties, and their security is paramount in the contemporary digital landscape. The process data from an individual (i.e., the revision history) can be at least as important as the outcome. However, process data tends to be much more sensitive and personal than the final outcome. Such data could be tampered with or exchanged with malicious intent. For instance, if trained on process data, deep learning methods could potentially generate “deep process fakes,” which can have disastrous and unintended consequences. The deep fake scenarios [34] have brought out the ability to alter photos and videos to the point of misrepresenting a person’s political stance, corporate identity, or involvement in a crime. Hence, securing and sharing personal process data with only trusted parties is critical. It is also urgent to develop strict policies and restrictions for machine or deep learning algorithms that are applied to individual process data. The current version of the Process Visualization techniques proposed in this work, including the n-gram similarity calculations, does not use any form of machine learning or deep learning.

4.4 Are learners monitored or controlled during their creative process?

The current release of our web application does not save any user data to our online servers, except at required times such as the time of loading the single-page web application, downloading a task description (question), explicitly saving a question or a response to the online storage server, verifying a human user, or when executing a computer code (to compile). However, any third-party components used could be using their own data collection techniques. All revision histories of a user's data are saved locally in the user's Internet browser's local storage. In other words, users have complete control over what process data they want to share and when. For instance, the web application does not track if a learner working on a project clears history, starts over, or abandons the writing process altogether.

The web application can be used entirely as a self-improvement tool. If learners wish to share their process, they can self-review it first and then decide if they want to share it. The current version of the application does not have any authentication built-in (e.g., logging in or signing in). The intent here is to make the website highly accessible to users since requiring an email address and forgetting passwords won't be a concern. Hence, users are not required to include any private information. *If a user writes their content, views the Process Visualizations, and downloads the data without saving it to our servers, our application has no knowledge of any such process and does not attempt to track them.* In sum, users have a clear choice to discontinue revision logging at any time, and if they do decide to share, can review their process data first on their own.

If the technique of maintaining a revision history was to be used in non-transparent ways (i.e., in exams, tests, or standardized language/programming tests), learners could feel vulnerable, monitored, and even controlled. This type of non-transparency could cause learners to resist using any education/learning system if given a choice, including the system presented herein. As an alternative, in a setting where learners feel monitored or controlled, an educator can motivate their learners by showing (modeling) their creative process using visualizations. For instance, an educator teaching writing can demonstrate how common it is to delete text during a revision process and an educator teaching programming could demonstrate how common it is to run into errors and spend time debugging.

4.5 Will educators' assessment time increase?

One concern that may be raised when exploring and interpreting Process Visualizations is that educators' assessment time could increase. It can be argued that educators worldwide are already struggling and need more time to provide feedback that promotes learning. Moreover, educators are increasingly concerned about how to build students' self-efficacy [35]. Unfortunately, finding time to develop tools that can help build students' self-efficacy and promote learning is a hurdle not many educators can achieve. Teachers and students only have so many hours each day, which is usually strictly regimented. Additionally, teachers cannot possibly know all the individual problems students have had in their learning process. At times, not even the student knows the problems that are causing barriers to learning. How, then, will educators be able to dig into the process of learning and providing feedback on the process? This is a genuine initial concern; however, several premises must be considered.

First, compared to reading an outcome (essay or code), visualizations can be faster to interpret, particularly when the structure of the visualization is consistent, with only the data changing. The first few exploratory activities and interactions will take time. However, once the structure and features of the visualizations are learned, the focus will naturally shift to seeing the story being unveiled by the data. These visualizations, however, should be improved for the highest accessibility and user-friendliness.

Second, after exploring a few varieties that may be possible within a visualization structure, users will develop knowledge about the range of possible variations and gradually become efficient at interpreting meanings from the visualizations. Short and dedicated tutorial videos on each visualization, aimed at enhancing visualization literacy, can also help the users learn how to interact with the visualizations.

Third, not all learners and educators should use the visualization toolbox for every task. Learners may only use it occasionally to learn about their style or with their peers to learn from each other. Similarly, educators

may use it only for a specific task or for a learner who would benefit from such feedback.

Fourth, in situations where the outcome is more important than the process, learners and educators can discard the Process Visualization approach altogether because these visualizations do not replace the outcome; they only supplement it. The premise of this research is that if learners and educators appropriately divide their time between reading the completed writing or coding tasks and exploring the Process Visualizations (i.e., consider both: the outcome and the process), the two can complement each other and together serve as a powerful resource to either self-learn or acquire feedback from others.

5 Conclusion

This work introduced several Process Visualizations aimed at facilitating writers and programmers to acquire feedback on their processes. The visualizations can encourage learners and educators to focus on the process and serve as tools to analyze their creative and cognitive processes. These visualizations and ideas can also serve as an inspiration for designing similar tools for learners and practitioners outside of writing and coding. For example, designers, music composers, artists, filmmakers, and architects could benefit from similar process-focused visualizations and analysis. Facilitating a focus on creative endeavor processes can also strengthen the core of current education systems, allowing learners to thrive. Such a process elucidating tool can also be helpful for special educators to collect data and write goals in the individualized education plans (IEPs) for differently-abled learners. Meanwhile, Process Visualizations and feedback may emerge as a new interdisciplinary field at the crossroads of psychology, education, computer science, and data visualization and can be quickly adopted in several existing systems and institutions throughout the future. For this to be effective and successful, the security of personal process data is also paramount.

Availability

The Process Feedback website is ready to be tested by educators and learners and is publicly available to all at www.processfeedback.org.

Acknowledgments

I am deeply grateful to the individuals who have contributed to the development of this manuscript. Their expertise and thoughtful insights have been essential in shaping and refining the content. Specifically, I extend my heartfelt appreciation to Dr. Shea Kerkhoff, Dr. Abderrahmen Mtibaa, Dr. Sambriddhi Mainali, Jason Wagstaff, and graduate students Shaney Flores and Kate Arendes at the University of Missouri-St. Louis, as well as undergraduate student Nilima Kafle, for engaging in several stimulating discussions and for dedicating their time to providing insightful feedback on the manuscript. Furthermore, I would like to acknowledge Dr. Manu Bhandari and Dr. Khem Aryal at Arkansas State University, Dr. Jie Hou from Saint Louis University, as well as Bishal Shrestha and Nitesh Kafle from Kathmandu, Nepal, for their valuable discussions and comments on the manuscript. I am also grateful for the recommendations provided by Dr. Amber Burgett at the University of Missouri-St. Louis and Dr. Laura March at the University of Missouri-Columbia. Finally, I am grateful to Carla Roberts at Preferred Copy Editing for editing the manuscript at a short notice.

References

- [1] Judith A Langer and Arthur N Applebee. *How Writing Shapes Thinking: A Study of Teaching and Learning*. NCTE Research Report No. 22. ERIC, 1987.
- [2] Perry D Klein and Pietro Boscolo. Trends in research on writing as a learning activity. *Journal of writing research*, 7(3):311–350, 2016.
- [3] Anne Whitney, Sheridan Blau, Alison Bright, Rosemary Cabe, Tim Dewar, Jason Levin, Roseanne Macias, and Paul Rogers. Beyond strategies: Teacher practice, writing process, and the influence of inquiry. *English Education*, 40(3):201–230, 2008.
- [4] Janet Emig. Writing as a mode of learning. *College composition and communication*, 28(2):122–128, 1977.
- [5] Roy D Pea and D Midian Kurland. On the cognitive effects of learning computer programming. *New ideas in psychology*, 2(2):137–168, 1984.
- [6] D Midian Kurland, Roy D Pea, Catherine Clement, and Ronald Mawby. A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, 2(4):429–458, 1986.
- [7] Shahira Popat and Louise Starkey. Learning to code or coding to learn? a systematic review. *Computers & Education*, 128:365–376, 2019.
- [8] John H Flavell. Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American psychologist*, 34(10):906, 1979.
- [9] Petros Georgiades. From the general to the situated: Three decades of metacognition. *International journal of science education*, 26(3):365–383, 2004.
- [10] National Research Council et al. *How people learn: Brain, mind, experience, and school: Expanded edition*. National Academies Press, 2000.
- [11] Cynthia J Brame. *Science teaching essentials: short guides to good practice*. Academic Press, 2019.

- [12] Emilie Hancock and Gulden Karakok. Supporting the development of process-focused metacognition during problem-solving. *PRIMUS*, 31(8):837–854, 2021.
- [13] Elke Baten, Magda Praet, and Annemie Desoete. The relevance and efficacy of metacognition for instructional design in the domain of mathematics. *ZDM*, 49:613–623, 2017.
- [14] Judith A Langer. *Children reading and writing: Structures and strategies*. Ablex Publishing, 1986.
- [15] Hakan Erdogmus, Maurizio Morisio, and Marco Torchiano. On the effectiveness of the test-first approach to programming. *IEEE Transactions on software Engineering*, 31(3):226–237, 2005.
- [16] John Hattie and Helen Timperley. The power of feedback. *Review of educational research*, 77(1):81–112, 2007.
- [17] Anastasiya A Lipnevich and Jeffrey K Smith. Effects of differential feedback on students’ examination performance. *Journal of Experimental Psychology: Applied*, 15(4):319, 2009.
- [18] Yun Xiao. The effects of training in peer assessment on university students’ writing performance and peer assessment quality in an online environment. *Doctor of Philosophy (PhD), Dissertation, Teaching and Learning, Old Dominion University*, 2010.
- [19] Kathleen M Cauley and James H McMillan. Formative assessment techniques to support student motivation and achievement. *The clearing house: A journal of educational strategies, issues and ideas*, 83(1):1–6, 2010.
- [20] James H McMillan and Jessica Hearn. Student self-assessment: The key to stronger student motivation and higher achievement. *Educational horizons*, 87(1):40–49, 2008.
- [21] Kathy Paulson Gjerde, Margaret Y Padgett, and Deborah Skinner. The impact of process vs. outcome feedback on student performance and perceptions. *Journal of Learning in Higher Education*, 13(1):73–82, 2017.
- [22] Paul Black and Dylan Wiliam. Assessment and classroom learning. *Assessment in Education: principles, policy & practice*, 5(1):7–74, 1998.
- [23] Lorrie A Shepard. The role of assessment in a learning culture. *Educational researcher*, 29(7):4–14, 2000.
- [24] Mantz Yorke. Formative assessment in higher education: Moves towards theory and the enhancement of pedagogic practice. *Higher education*, 45:477–501, 2003.
- [25] Dante D Dixon and Frank C Worrell. Formative and summative assessment in the classroom. *Theory into practice*, 55(2):153–159, 2016.
- [26] David Boud. Assessment and learning: contradictory or complementary. *Assessment for learning in higher education*, pages 35–48, 1995.
- [27] Marc Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.
- [28] Eugene W Myers. An o (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.
- [29] Herman Zvonimir Došilović and Igor Mekterović. Robust and scalable online code execution system. In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1627–1632, 2020.
- [30] Satrajit S Ghosh, Arno Klein, Brian Avants, and K Jarrod Millman. Learning from open source software projects to improve scientific review. *Frontiers in computational neuroscience*, 6:18, 2012.
- [31] Felice Frankel and Angela H DePace. *Visual strategies: A practical guide to graphics for scientists & engineers*. Yale University Press, 2012.

- [32] Edward R Tufte. *Beautiful evidence*, volume 1. Graphics press Cheshire, CT, 2006.
- [33] Alberto Cairo. *The truthful art: Data, charts, and maps for communication*. New Riders, 2016.
- [34] Bobby Chesney and Danielle Citron. Deep fakes: A looming challenge for privacy, democracy, and national security. *Calif. L. Rev.*, 107:1753, 2019.
- [35] David J Nicol and Debra Macfarlane-Dick. Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in higher education*, 31(2):199–218, 2006.