FFL: A Language and Live Runtime for Styling and Labeling Typeset Math Formulas

Zhiyuan Wu
wuzed@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Jiening Li
jiening@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Kevin Ma
makev@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Hita Kambhamettu
hitakam@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Andrew Head
head@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

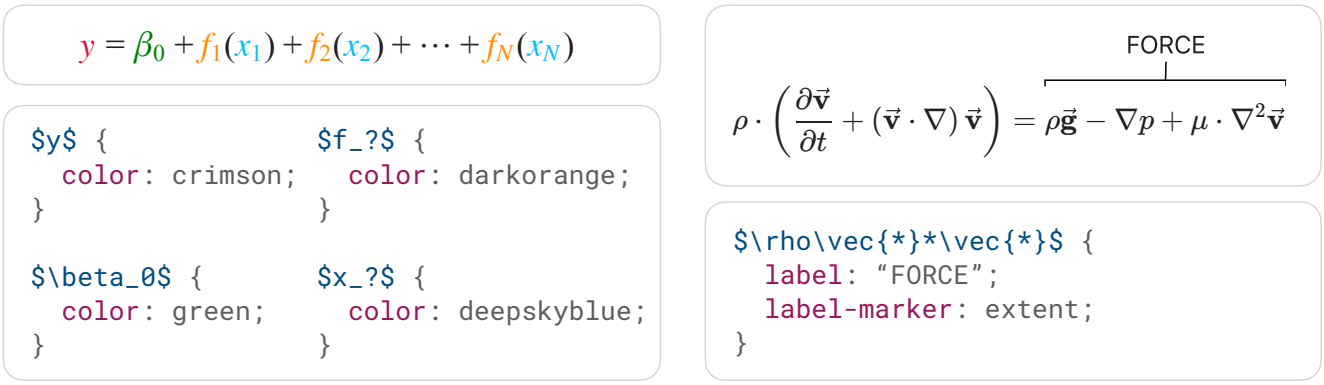


Figure 1: Two augmented formulas and their accompanying augmentation specifications written in FFL ("Formula Formatting Language"). FFL is designed to be concise, writable, readable, and integrable into web-based document authoring environments. Augmentations are specified using selectors (dark blue) that match classes of expressions, and properties (magenta) that apply augmentations like color and labels to formulas. The language can be processed with its live runtime offering rapid feedback to notation authors. The pictured augmentations are adapted from those in documents by Hohman et al. [28] and Murad [56].

ABSTRACT

As interest grows in learning math concepts in fields like data science and machine learning, it is becoming more important to help broad audiences engage with math notation. In this paper, we explore how authoring tools can help authors better style and label formulas to support their readability. We introduce a markup language for augmenting formulas called *FFL*, or "Formula Formatting Language," which aims to lower the threshold to stylize and diagram formulas. The language is designed to be concise, writable, readable, and integrable into web-based document authoring environments. It was developed with an accompanying runtime that supports live application of augmentations to formulas. Our lab study shows that FFL improves the speed and ease of editing augmentation markup, and the readability of augmentation markup compared to baseline LaTeX tools. These results clarify the role tooling can play in supporting the explanation of math notation.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**.

KEYWORDS

formulas, interactive typesetting, liveness, labels, colors

ACM Reference Format:
Zhiyuan Wu, Jiening Li, Kevin Ma, Hita Kambhamettu, and Andrew Head. 2023. FFL: A Language and Live Runtime for Styling and Labeling Typeset Math Formulas. In *The 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23), October 29-November 1, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3586183.3606731

UIST '23, October 29-November 1, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0132-0/23/10.
https://doi.org/10.1145/3586183.3606731

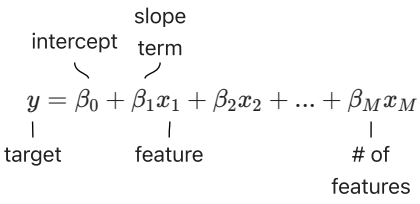
1 INTRODUCTION

Notation poses a barrier to understanding mathematical ideas. Whether in the physics classroom, data science research papers [57], or programming documentation [7], readers find important knowledge locked behind the formalisms of formulas and symbols. Consider a reader encountering this formula in a research paper [28]:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_M x_M$$

This formula represents a linear regression model. If the reader is not familiar with its idioms, they are likely to find it hard to understand. For instance, what is "x" and how is it different from "β"? β_0 and β_1 share a common base—how are they related? What is the intuition of the formula as a whole?

Suppose the formula was instead shown as follows:



This alternative presentation helps a reader to unpack the meaning of a formula. It helps the reader understand the purpose of the formula as predicting a target value from a set of input features. It clarifies that "x" terms correspond to features, and "β" terms correspond to weights. And it brings the formula into a realm of familiarity by relating "β_0" and "β_1" to the ideas of intercept and slope terms that are taught in algebra class. Annotated formulas like these help readers grasp their meaning at glance. The annotations' value becomes particularly pronounced when applied to formulas of yet greater complexity and domain specificity.

In this paper, we seek to advance the state of the art in tooling that allows authors to create augmented formulas like these. A recent survey by Head et al. [27] reveals the challenges present in building effective interactive tooling for this purpose. To start with, conventional formula typesetting tools make it a "struggle" to augment formulas. Formula markup gets too messy, and environments provide insufficient support for experimenting with cross-document formula styling choices.

Our contribution is a reinvention of the process of augmenting formulas in typesetting tools. We envision formula augmentation as a process that involves a crisp markup language and live incremental feedback. We reify this vision in *FFL*, or "Formula Formatting Language," a markup language for formula augmentation. FFL is targeted for web-based math document authoring. Its key innovations are a design that splits augmentation markup from formula markup, a CSS-inspired familiar syntax, support for cross-document styling, and an implementation that permits live feedback.

We assess FFL's impact on the authoring experience in a controlled usability study where 28 participants used FFL and a LaTeX baseline. In complex editing tasks, FFL increased efficiency, self-reported ease, and led to more readable augmentation code versus the baseline. For tasks involving writing simple augmentations from scratch, FFL and LaTeX saw no significant differences. Reviewing the evidence in the framework of the cognitive dimensions of notation [4], our study suggests FFL reduces viscosity, hard mental operations, and error proneness, while benefiting from closeness of mapping and progressive evaluation. These results suggest that FFL-like languages could make the formula augmentation task better supported in contemporary authoring tools.

In summary, this paper contributes:

- The design of *FFL*, a markup language for augmenting formulas, designed for readability and efficiency.
- A runtime supporting live application of augmentations to formulas in web-based authoring environments.
- Evidence from a usability study that FFL leads to faster and easier edits to augmentation markup, and results in more readable markup.

2 BACKGROUND AND RELATED WORK

In this section, we discuss why authors might wish to augment notation and then situate our work relative to prior tools for authoring and augmenting formulas.

2.1 Notation and augmentation

Math notation is difficult to read. It has been described as a language of its own, requiring practice to understand [1]. Over time, the human perceptual system can become trained to recognize structures in formulas [47]. Readers learn idioms in formulas through repeated exposure, such that experts can spot structures in formulas novices miss [66]. For novices, notation poses a barrier to understanding mathematical texts and is often cited as a challenge in self-teaching machine learning [7] and reading research papers [57].

Subtle changes to the presentation of notation can affect its readability. For instance, coloring and annotating formulas can reduce cognitive load in solving algebra problems [77]. Readers can be aided in understanding operator precedence by altering which letters are used for variables and spacing between variables [22, 25]. The design space for augmented notation is large. Dragunov and Herlocker [16] propose augmenting formulas with symbols definitions, annotations that show how variables are manipulated across stages of derivation, and controls that adjust the level of detail in a derivation. Head et al. [27] and Hohman et al. [29] expand this design space, with the former describing 16 classes of augmentations. In this paper, we explore how authoring environments could be extended to equip authors with tools to perform common some of the most common kinds of augmentations.

2.2 Tools for augmenting notation

Markup languages. One of the most common kinds of tools for writing and augmenting notation is the markup language. Markup languages, like TeX [35], allow authors to write formulas in plain text and render them as cleanly typeset formulas. Some such tools provide support for augmentation. LaTeX [68], for instance, supports the addition of color with the `color` [8] package, and labels with macros from the `mathtools` [45] and `annotate-equations` [32] packages. Recent research suggests that these tools could benefit from cleaner markup design, better defaults, and better support for cross-document style changes [27].

The popularity of TeX as a language for formula typesetting has led to web-based TeX formula typesetters. One such tool is KaTeX [17]. The context of the web provides new opportunities

for augmentation. KaTeX offers authors the `\htmlClass` for assigning HTML classes to arbitrary expressions. CSS can then be used to apply styles to those expressions. Our goal with FFL was to support augmentation of TeX formulas in web-based authoring environments using a language similar to CSS.

Notation augmentation is a feature of several recent markup languages for math and science communication. Nota [13] and Heartdown [41] let authors specify definitions of symbols, revealing symbol definitions in the margins of selected formulas when clicked upon. Curvenote's editor API [14] provides support for parametric LaTeX formulas, where numeric values can be substituted into formulas as users interact with widgets. *manim* [62] supports the creation of animations of math formulas with step-by-step builds and incremental annotation. We share motivation with these projects, aiming to create an extensible augmentation language and runtime for static math texts [27].

Why focus on augmentation in markup editors rather than other sorts of document editors? Markup, and in particular TeX, is used pervasively within the sciences and academia. It is a preferred tool for writing math notation not necessarily for sketching out ideas, but rather for disseminating or archiving them [54]. One study suggests writers can enter notation-dense passages more efficiently with TeX than with structured editors [34]. TeX is used widely enough that WYSIWYG editors like Word have incorporated it as a language for formula input [49]. We see the development of effective markup-based augmentation tools as a natural springboard for efforts to develop better augmentation tools generally.

Structured editors. WYSIWYG document editors like Word [51] sometimes provide structured formula editors. These editors can be used to augment formulas by selecting labels from menus, or by applying their tools for formatting text. Toolkits like MathType make such functionality available as a plugin to other editing applications [70]. One advantage of these tools is that their WYSIWYG design makes augmentation affordances easier to discover.

Vector graphics editors. Formulas can be augmented using vector-based graphics editing software; Head et al.'s study describe Google Slides [23], Inkscape [31], Mathcha [48], and PowerPoint [52] as several tools that authors are already using. Some of these tools require authors to render formulas outside of the environment (e.g., with CodeCogs [10]) and import the render as a bitmap or vector graphics into the editor. These tools are often both familiar to authors and flexible—authors can add augmentations using the full complement of text formatting and shapes the tools provide. FFL and vector graphics editors occupy two complementary areas of the augmentation design space, with FFL focusing on supporting typesetting experience and transferable styles, and vector graphics editors offering flexible augmentation through direct manipulation.

Sketch and gesture. Formulas can also be written and augmented as sketches [38, 39, 63, 78]. In sketching tools, augmentations are naturally supported when authors are given the ability to change ink color and draw free-form shapes. Some sketching tools support unique augmentations, like linking expressions to sketched physical objects [38, 63], or manipulating expressions with gestures [50, 73, 78]. Some of these affordances could be adapted as advanced augmentations for languages like FFL in the future.

Automation. As text understanding techniques improve, it may be possible to automatically augment notation. Myriad projects have explored the ability to detect the positions of symbols [26] and parse formulas [2, 44] from arbitrary input documents. Should it become possible to reliably detect expressions and their meaning automatically, augmentations could be added to documents with reduced input from authors.

2.3 Tools for augmenting texts

Our work draws inspiration from HCI research that develops powerful text authoring affordances generally.

Repetitive text editing. One challenge in editing longer texts is making repetitive edits when revising repeated phrases and ideas. HCI research has proposed numerous techniques to do so, including linked editing [69], detection and propagation of edits [53, 58], and editable macros [24]. FFL's approach is to allow authors to use CSS-style selectors to indicate which expressions to augment. These selectors allow authors to apply and edit augmentations for many related expressions at once.

Diagrams. One of the facilities of FFL is to support the creation of simple formula diagrams where descriptive labels are linked to expressions. Researchers have developed powerful domain-specific languages supporting for diagramming like Penrose [76] and Bluefish [59]. In comparison to these prior toolkits, the aim of FFL's labeling system is to support ease and conciseness in supporting a common, simple sort of labeling, among other augmentations.

Live feedback. FFL supports third-level or "edit-triggered" liveness, according to Tanimoto's taxonomy of liveness [67]. Liveness has been a central feature of dozens of research systems [60]. Its use in LaTeX tooling (e.g., [15, 21]) may arise from the fact that LaTeX documents require time-consuming compilation to view the effect of a change. FFL incorporates liveness to equip authors with more rapid feedback as they are experimenting with augmentations.

3 DEMO

FFL is designed to help authors augment formulas with a lightweight syntax and live feedback. Here, we illustrate the envisioned user experience of FFL with a scenario.

Imagine Auggie, a researcher writing an article in a web-based scientific authoring environment. They are writing a passage where they introduce the idea of linear regression.[1] They wish to help readers understand the gist of this formula, despite the dense appearance of the formula and the accompanying prose:

[1]This example is adapted from an excerpt from [28, Hohman et al.].

Consider a dataset $D = \{(x_i, y_i)\}^N$ of N data points, where $x_i = (x_{i1}, x_{i2}, \cdots, x_{iM})$ is a feature vector with M features, and y_i is the target, i.e., the response, variable. Let x_j denote the jth variable in feature space. A typical linear regression model can then be expressed mathematically as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$

This model assumes that the relationships between the target variable y_i and features x_j are linear and can be captured in slope terms $\beta_1, \beta_2, \ldots, \beta_M$.

Auggie desires to augment the formula using colors and labels to expose the formula's meaning. Their editor has been extended with support for FFL, which allows them to experiment with these augmentations. Auggie first explores how they could use of color to help readers correlate expressions in the formulas with their descriptions in the text.

To start, Auggie colors the target variable y. To do this, they write the following FFL selector and style in a text editor adjacent to their document markup. This helps ensure that the augmentation markup does not clutter the formula or document markup.

```
$y$ { color: red }
```

This markup represents a request to find all instances of symbols described by the LaTeX literal "y" and color them red. The effect is instantaneous: as soon as Auggie finishes typing "`red`," the symbol y is colored red everywhere it appears in the document:

Consider a dataset $D = \{(x_i, y_i)\}^N$ of N data points, where $x_i = (x_{i1}, x_{i2}, \cdots, x_{iM})$ is a feature vector with M features, and y_i is the target, i.e., the response, variable. Let x_j denote the jth variable in feature space. A typical linear regression model can then be expressed mathematically as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$

This model assumes that the relationships between the target variable y_i and features x_j are linear and can be captured in slope terms $\beta_1, \beta_2, \ldots, \beta_M$.

The next step is to use color to help readers find the description of y in the text. As Auggie is writing in a web-based environment, they can mix in some CSS to format the text. The CSS can be written alongside the FFL.

To style the text, Auggie surrounds the definition phrase with a `span` tag and gives it the class "`target`." Then they give the definition the same color by adding a selector for the span, "`*.target`", next to the FFL selector.

```
$y$, *.target { color: red }
```

Consider a dataset $D = \{(x_i, y_i)\}^N$ of N data points, where $x_i = (x_{i1}, x_{i2}, \cdots, x_{iM})$ is a feature vector with M features, and y_i is the target, i.e., the response, variable. Let x_j denote the jth variable in feature space. A typical linear regression model can then be expressed mathematically as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$

This model assumes that the relationships between the target variable y_i and features x_j are linear and can be captured in slope terms $\beta_1, \beta_2, \ldots, \beta_M$.

Auggie is not content with the augmentation, wishing to try out other, less harsh colors. As they experiment with other colors from `DarkRed` to `Crimson`, they see the visual effect live, receiving the rapid feedback common to online Markdown editors, but less common to LaTeX document editors that require recompilation.

```
$y$, *.target { color: DarkRedCrimson }
```

Consider a dataset $D = \{(x_i, y_i)\}^N$ of N data points, where $x_i = (x_{i1}, x_{i2}, \cdots, x_{iM})$ is a feature vector with M features, and y_i is the target, i.e., the response, variable. Let x_j denote the jth variable in feature space. A typical linear regression model can then be expressed mathematically as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$

This model assumes that the relationships between the target variable y_i and features x_j are linear and can be captured in slope terms $\beta_1, \beta_2, \ldots, \beta_M$.

Now that Auggie is content with the colors they chose, they notice that they wish for the subscripts of y expressions to be colored as well. To augment all y expressions with subscripts, Auggie only needs to make a small edit. They add "`$y_*$`" to the list of selectors, and see the crimson color applied to the intended expressions.

```
$y$, $y_*$, *.target { color: Crimson }
```

Consider a dataset $D = \{(x_i, y_i)\}^N$ of N data points, where $x_i = (x_{i1}, x_{i2}, \cdots, x_{iM})$ is a feature vector with M features, and y_i is the target, i.e., the response, variable. Let x_j denote the jth variable in feature space. A typical linear regression model can then be expressed mathematically as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$

This model assumes that the relationships between the target variable y_i and features x_j are linear and can be captured in slope terms $\beta_1, \beta_2, \ldots, \beta_M$.

The next step is to help readers understand the other major expressions in the formula, namely the x's (blue) and β's (purple). Auggie decides to assign each a distinct color that will help a reader look up the respective definitions in the text. To do so, they create a similar style block for each group of variables they wish to augment:

```
$y_?$, $y$, *.target { color: Crimson      }
$x_*$, *.feat        { color: DodgerBlue   }
$\beta_*$, *.slope   { color: MediumPurple }
```

Consider a dataset $D = \{(x_i, y_i)\}^N$ of N data points, where $x_i = (x_{i1}, x_{i2}, \cdots, x_{iM})$ is a feature vector with M features, and y_i is the target, i.e., the response, variable. Let x_j denote the jth variable in feature space. A typical linear regression model can then be expressed mathematically as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$

This model assumes that the relationships between the target variable y_i and features x_j are linear and can be captured in slope terms $\beta_1, \beta_2, \ldots, \beta_M$.

After inspecting the augmented passage, Auggie wishes that β_0 was not given the same color as the other β terms, because it is better described as an intercept rather than a slope term. They revert the style for just β_0 by adding an additional one-line rule, setting the color of β_0 to `inherit`, as one might do in CSS, rather than accept the color of the other β terms.

```
$y_?$, $y$, *.target { color: Crimson      }
$x_*$, *.feat        { color: DodgerBlue   }
$\beta_*$, *.slope   { color: MediumPurple }
$\beta_0$            { color: inherit      }
```

Consider a dataset $D = \{(x_i, y_i)\}^N$ of N data points, where $x_i = (x_{i1}, x_{i2}, \cdots, x_{iM})$ is a feature vector with M features, and y_i is the target, i.e., the response, variable. Let x_j denote the jth variable in feature space. A typical linear regression model can then be expressed mathematically as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$

This model assumes that the relationships between the target variable y_i and features x_j are linear and can be captured in slope terms $\beta_1, \beta_2, \ldots, \beta_M$.

Auggie is satisfied with this result. Throughout their exploration, FFL provided a lightweight syntax for making cross-cutting notation augmentations with live feedback.

Further design space exploration. There is more than one way to augment a formula to expose its meaning. Auggie considers another strategy that they think will make their article more skimmable which relies less on the textual description (omitted below), and instead exposes descriptions of expressions in labels. FFL helps them experiment with this style of augmentation as well.

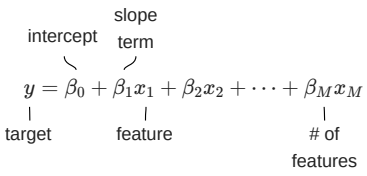
Auggie starts from a fresh FFL style sheet, this time adding augmentations in the form of labels. They first add a label for y, describing it as the "target" of prediction.

```
$y$ { label: target }
```

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_M x_M$$
target

They then add labels for the remaining expressions. This is a matter of adding one style block per annotated expression.

```
$y$              { label: target     }
$\beta_0$        { label: intercept  }
$\beta_?$:nth(1) { label: slope term }
$x_1$ {
  label: feature;
  label-position: below
}
$M$ { label: # of features }
```



As Auggie does so, labels render live. The labels are automatically laid out to reduce overlap and maximize adjacency of labels to expressions. In this way, Auggie can think about augmentations at a high level, avoiding the work of manually arranging labels. Notably the labels are tolerant to future changes to the formula: should Auggie add additional β and x terms to the formula, the labels will move as the formula adjusts its position.

When they are finished with this document, Auggie could save their style sheet for use in other documents with notation that deserved to be described in similar ways.

4 SYSTEM

In the section, we describe FFL, a language and live runtime for augmenting typeset math formulas in web documents. FFL was designed and developed following an iterative approach. Fine-grained decisions about syntax design were informed by pilot usability studies with early versions of the tool.

Acknowledging the challenges of writing augmentation markup revealed in prior work [27], the goals of FFL were as follows:

- Basic augmentations should be easy to read and write;
- Authors should receive rapid feedback on their designs;
- Augmentations should be aesthetically pleasing;
- Authors should be supported to experiment with cross-cutting augmentation choices.

Below, we describe the two main components of the FFL toolkit: the language, and the supporting live runtime.

4.1 Language design

The FFL language is a CSS-like language for specifying augmentations for formulas. FFL was designed to resemble CSS due to the latter's use as a separable styling language in web authoring environments. We envisioned authoring environments where eventually authors write FFL and CSS side-by-side.

```
$x_i$, ... {
   color: red;
   ...
}
```

Like CSS, FFL in essence consists of declarations of style rules. Each style rule block consists of a selector indicating what expressions the augmentation applies to, and a set of property declarations describing augmentations to apply, resembling the inset figure.

One advantage of this format is that FFL can be easily transpiled to CSS for a myriad of simple styles (e.g., color, font weight). Below demonstrates the current expressive potential of FFL's syntax. A visual summary of language constructs appears in Figure 2. Our focus is on describing the language primitives, and the augmentations we have built into the language to date. We intend the language to be further extended to support additional augmentations.

4.1.1 Selections. An author conveys which math expressions to augment by writing *selectors*. FFL provides a flexible selector syntax, allowing for literal matches to LaTeX substrings, wildcards, predefined classes, and combinators.

Literal Selectors. The simplest way to select an expression is to write the LaTeX for the expression one wishes to augment. Writing a literal selector entails writing a LaTeX string, with its typical (`$`) delimiters on either side. Literal selectors are resilient to some simple variations in how an expression might be written in LaTeX: for instance, the selector "`$x_i$`" matches the expression x_i regardless of whether it is written "`$x_i$`" and "`$x_{i}$`".

Wildcards. Authors can select syntactically related expressions using wildcards. Two kinds of wildcards are provided, inspired by the `glob` [37] wildcard syntax used in Unix command lines. *Character wildcards* match single characters, and are written "`?`". For instance, "`$x_?$`" selects all symbols that have x as a base and a single character as subscripts. *Sequence wildcards* match strings of unbounded length, and are written "`*`". For instance, "`$f(*)$`" selects $f()$, $f(0)$, $f(x)$, and $f(x+1)$, among other expressions. Authors can match the literal characters "`?`" and "`*`" by escaping them with a backslash (i.e., as "`\?`" and "`\*`").

Expression classes. FFL provides classes for common categories of expressions, such as superscripts, subscripts, and constants. All classes are preceded by dots ("`.`"), like typical CSS classes.

Class	Matches
`.constant`	$0, 1, \cdots$
`.superscript`	$x^0, e^{i\pi}, \cdots$
`.subscript`	$m_0, n_k, \cdots$
`.numerator`	$\frac{1}{2}, \frac{k}{m}, \cdots$
`.denominator`	$\frac{1}{2}, \frac{k}{m}, \cdots$

Supported classes are shown in the inset figure. These classes become particularly powerful when used within combinators, permitting an author to select, for instance, squares as the appearance of the literal "2" within superscripts.

Indexed groups. To disambiguate between selections, we offer another special class named "`.group`", referring to portions of the formula markup surrounded by double braces (e.g. `{{...}}`). The modifier "`:nth(`*i*`)`" can be appended to any selector to select the *i*-th matching expression. Authors select a specific group by using the modifier in conjunction with the "`.group`" selector.

Combinators. Selectors can be composed to make them more general or more precise. Selections can be made more precise with the intersection combinator, "`intersection(`*selector1, selector2, ...*`)`," which selects expressions matching all selectors provided as arguments. A shorthand for intersection is provided as "`selector1 selector2 ...`," which is reminiscent of CSS's compound selectors; with this shorthand authors can express intersections as if they were selecting `selector2` from within `selector1`. The union combinator, as with CSS, uses a comma ("`,`") to separate selectors, matching any expression that matches one of the selectors.

CSS Selectors. To select HTML elements from within an FFL style specification, an author can prepend an asterisk ("`*`") to the name of a class (e.g., "`*.cls0`").

4.1.2 Augmentations. The FFL language supports specification of two kinds of augmentations: styles and labels. Permitted augmentations include color and labels, the two most commonly used kinds of augmentations according to a recent survey [27].[2]

Style. Styles are alterations to the expression elements, like color, font weight, and background. They correspond roughly to CSS properties, and share the same names (e.g., "`color`," "`font-weight`"). Because these properties are transpiled into CSS, they accept all of the same property values as CSS (e.g., colors can be specified using HTML color codes, hex codes, "`rgba(...)`" values). As described in Section 5.4, some styles require additional processing on the backend to provide the expected styling behavior in the unique setting of HTML typeset math formulas.

Labels. FFL provides language primitives for creating and customizing labels that describe expressions. The "`label`" property allows an author to define and show a label for an expression: upon specifying this property, a label will appear next to the first appearance of that expression in a formula, connected to that expression with a leader line. The "`label-marker`" property allows the author to specify what kind of marker should connect the label to the expression. The marker can be either a `leader` line or an `extent` marker, i.e., a bracket shown in the margin; extent markers are particularly useful for labeling long expressions.

Label placement is automatic, and is designed to avoid overlapping labels and to place labels as close to their corresponding expressions as possible. A label is applied only once to any given formula; it is anchored to the first appearance of the labeled expression. Should an author wish to customize the placement of labels, they

[2] An analysis of the spreadsheet in Head et al.'s [27] supplemental material shows 69% of augmented formulas in their sample made use of either font color or labels.

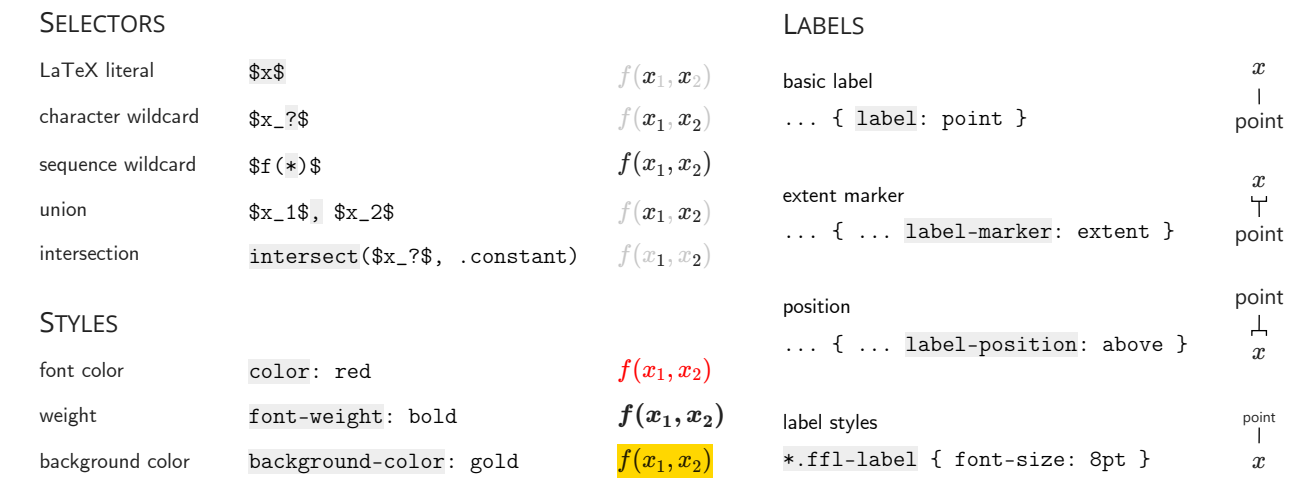


Figure 2: A visual specification of the FFL language, including its constructs for selecting expressions, styling, and labeling formulas. Each row names a language feature, provides an example snippet of FFL, and shows the result of its application to one of the example formulas $f(x_1, x_2)$ or x.

may do so by defining the "label-position" property to place the label either "above" or "below" the formula. To support further label customization, all labels allow values of "html(...)" (sanitized for security by default), or are generated as HTML text spans with the class "ffl-label" class. In this way, their appearance (e.g., font size, font family, color) can be configured with normal CSS properties by using the CSS selector "*.ffl-label."

4.2 Live runtime

FFL was designed to be incorporated into arbitrary web-based text editing tools as a live styling utility, and for integration into articles generated from these editors. In this section, we describe how the runtime to support integration as a live styling tool.

4.2.1 FFL library. To ease the work involved in integration, FFL is implemented as a light wrapper around widely-used *KaTeX* [17] tool. KaTeX is a tool that typesets LaTeX formulas on web pages. It is used in a variety of web-based authoring tools, including Dropbox Paper, Observable, Gatsby, Messenger, and Quill. It is also one of the supported formula rendering engines in Jupyter Lab [33].

To integrate FFL into a web authoring environment, a developer would do the following. First, they would create editor widgets (like text areas) for authors to write FFL in. Second, they would replace calls to KaTeX's formula typesetter with a call to a nearly equivalent API on FFL. That method has the signature:

```
ffl.render(latex: string, ffl: string,
  renderTo: HTMLElement, options?: KatexOptions): void
```

where "latex" is the LaTeX markup for the formula, the "ffl" parameter takes in the FFL style specification, "renderTo" is a reference to the HTML element into which to render the augmented formula, and "options" is an object of KaTeX options for typesetting the formula. If called without a "renderTo" target, the method returns the HTML string for the rendered formula.

4.2.2 Supporting live evaluation. To support live evaluation of an FFL style specification in an editing environment, the one necessity is to trigger a new call to "ffl.render" whenever the LaTeX or the FFL specification changes. To demonstrate the feasibility of such an integration, we implemented a Markdown editing environment with live FFL integrated. To develop this environment, we first created a document editor as a simple text area. When authors write Markdown in the text area, the Markdown is passed to the *markdown-it* [12] open source Markdown parser and then rendered into a document view next to the Markdown editor.

We created a pluggable *markdown-it* extension to call the FFL API, rather than the KaTeX API, to render math formulas; the FFL API is called with an FFL style specification that authors write in another text box adjacent to the Markdown text box. Live evaluation is supported by triggering a parse of the Markdown when either the Markdown or the FFL specification is edited. A demo of the authoring environment appears in the accompanying video.

5 IMPLEMENTATION

The FFL runtime required an implementation that would translate specifications of augmentations to rendered HTML formulas. Figure 3 summarizes the translation process and the intermediate representations involved. Here, we briefly describe our implementation of augmentations in the FFL runtime in terms of each time the API is triggered.[3]

5.1 Parsing the FFL specification

FFL markup is parsed using a custom parser for the FFL grammar. The parser was generated by Peggy [46], a PEG parser generator, from an FFL grammar that resembles a subset of CSS grammar.

[3] Our implementation is hosted at penn-hci.github.io/ffl.

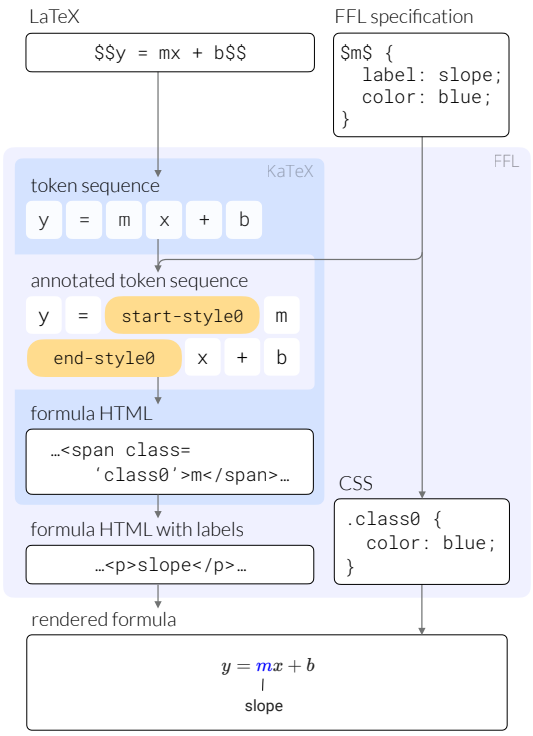


Figure 3: The generation of an augmented formula from LaTeX and an FFL style specification. FFL wraps the KaTeX library [17], shimming itself into KaTeX's token parsing to detect and annotate expressions of interest. KaTeX generates an annotated HTML formula, which can be styled with CSS that FFL generates from its specification. FFL augments the generated HTML with labels by post-processing the generated HTML.

5.2 Matching LaTeX token sequences

Next, the selectors are used to identify ranges of formula LaTeX that need to be augmented. To do this, we use KaTeX to lex both the selectors and the formula LaTeX into token sequences, with a small amount of parsing to normalize implicit groups. Then, we scan the LaTeX formula token stream for sub-sequences matching the selector, similarly tokenized by KaTeX. A segment and selector are considered matching if they contain a sequence of matching tokens. Literal tokens are considered matching if they are equal and wildcards ? and * match either a single or a sequence of tokens respectively. The current implementation of sub-sequence search permits matching overlapping sub-sequences, and wildcard matches for the character and sequence wildcards.

Once a matching sub-sequence is found, KaTeX must be told to augment the characters in that sub-sequence. To do this, we insert special tokens before and after the sub-sequence. These special tokens instruct KaTeX to insert temporary `span` tags with a generated class name around the expression in the rendered formula HTML. While it inserts these special tokens, FFL creates a map from FFL selectors to the selector-specific class names, from which it builds a CSS style sheet that applies FFL styles (e.g., color, font weight) to the expression in the rendered HTML formula.

We implement the search for matching sub-sequences of tokens in a way that does not require changing KaTeX's implementation. Our approach is to handle matching in a custom KaTeX macro that we wrap around each formula. With KaTeX, macros are defined as JavaScript functions. When KaTeX expands a macro, it does so by calling the corresponding JavaScript function, passing the function a sequence of tokens found in the macro's arguments. We wrote a custom macro that, when expanded by KaTeX, takes the tokens of the formula, searches for matching sub-sequences, modifies those sub-sequences as described in the paragraphs above, and returns the modified tokens to KaTeX for further processing.

5.3 Applying styles

Once KaTeX produces the HTML for a rendered formula, FFL traverses the HTML to associate styles with matched expressions. FFL searches for the previously inserted `span`s, removing them and applying generated CSS class names to the HTML elements between them. Then, it appends the generated CSS (including color and font weight) to a "`style`" element in the DOM, which has the effect of styling the matched elements in the expression.

5.4 Drawing overlays and underlays

Finally, all remaining kinds of augmentations collected during the HTML tree traversal are applied, including labels and background colors. Labels are drawn as relatively positioned HTML elements on the margins of the formula inside an SVG [72] element. Label positions are determined by locating the position and outer bounding box of all tokens in its corresponding expression. Label positions are adjusted to reduce overlap by Labella [36]. The formula is padded with additional space so that labels do not occlude the surrounding text. Background colors are implemented as relatively positioned boxes placed behind the corresponding expression; this implementation is necessary to support background colors for selections whose constituent elements have a joined area that differs from the rectangular bounding box of the whole expression to ensure that there is only one background box, rather than multiple overlapping boxes for each character element in the expression.

5.5 Technical limitations

Our current technical approach suffices for reifying the ideas behind FFL in a working tool. Here, we describe technical limitations that should be addressed to increase FFL's flexibility and robustness.

Behavior of sequence wildcard. One revelation from our development was that the `glob`-style "`*`" wildcard is not well-defined for strings with the inherent hierarchy of LaTeX formulas. The current behavior of "`*`" is to match any terminal token or group at the same group level as the "`*`" in the selector. This decision remains to be more closely examined.

Block styles. For some styles, FFL transpiles directly to CSS. For others like background color, border, and padding, FFL requires custom solutions. The default approach of FFL is to apply a style to all tokens separately in an expression. Rather, block styling augmentations—like padding—should apply to an expression in

whole. To overcome this brittleness for block styling, we believe future versions of FFL should use KaTeX's render to MathML [19] instead of HTML; MathML contains structures that can be more easily detected and augmented for these styles, and has become mainstream into most major browsers earlier this year.

Performance. Preliminary tests on a commodity laptop show that rendering a formula with FFL takes tens of milliseconds (i.e., 35ms to augment "x" in the linear regression formula from the demo section). This runtime is imperceptible for single augmentations applied to single formulas. Our tests lead us to attribute latency to the time FFL takes to insert augmentation markers into KaTeX's token stream: latency increases as more matches are found in the formula (e.g., it takes 50ms to augment expressions matching "`$*$`" in the same demo formula). As the number of augmentations and formulas grow, adjustments will be required (e.g., optimizations, parallelization) for FFL to continue to deliver instant feedback.

6 EVALUATION

To evaluate FFL's impact on the experience of authoring augmentations, we conducted an in-lab usability study. The study was designed to answer the following questions:

(1) How does FFL influence authors' ability to create and edit augmentations?
(2) How could tools like FFL be improved to better support formula augmentation?

The study consisted of a controlled comparison between FFL and a LaTeX baseline for augmentation creation and editing tasks, followed by an exploratory authoring task with FFL.

6.1 Participants

We sought participants with experience authoring math documents with LaTeX. Participants were recruited from graduate student mailing lists at a computer science program at a private university, with the sole prerequisite of prior experience writing LaTeX formulas.

33 participants were recruited in total. The vast majority were master's students; 7 were students in a joint bachelor's / master's program. 3 described themselves as software developers, 1 as an academic researcher, and 1 as a teacher.

Participants reported their prior experience with LaTeX as follows: 24% had less than 1 year of experience; 48% had 1–2 years, 21% had 3–5 years, and 6% had more than 5 years. 55% used LaTeX weekly, 18% monthly, and 24% less often than monthly. Participants reported their comfort with LaTeX as a median of 4 on a 5-point Likert scale ($\sigma = 0.8$, IQR = 1). They were considerably less comfortable with CSS, with a median comfort level of 2 out of 5 ($\sigma = 1.0$, IQR = 1).

6.2 Procedure

6.2.1 Setup. All study sessions were conducted in person in an HCI usability study lab. Participants completed tasks using a computer with a large external monitor, keyboard, and USB mouse. Progress was managed by a custom web app we built to facilitate the study. This app opened the user interfaces participants were expected to use for tasks, and pre-loaded them with task stimuli. It also opened questionnaires after each task. For FFL tasks, participants used a custom live editing environment. For LaTeX tasks, they used Overleaf [43]. Two participants needed to complete the tasks on a personal laptop instead of the lab computer; these participants' data were used in our qualitative analysis but omitted from the quantitative analysis (Section 6.3).

6.2.2 Tutorial. Participants were given 10-minute tutorials of how to augment formulas with both of the interfaces under study—FFL and the LaTeX baseline. A member of the research team demonstrated how to perform key augmentation actions, like selecting expressions, coloring them, and labeling them with line and extent labels, both above and below the formula. Tutorial materials were designed to maximize parity in how the interfaces were introduced while minimizing complexity of the learning material. Participants were asked to practice each feature that was introduced on a sample formula. They were provided with a cheat sheet for each interface to use as a reference during the tasks.

6.2.3 Interfaces. The two interfaces participants used were a live editor with FFL support, and a baseline LaTeX environment. The FFL environment is described in Section 4.2.2. The environment offers only limited support for error recovery: if the author enters a syntactically invalid specification, the interface reports that an error was found without reporting character positions, while continuing to show the render of the last correct specification. In LaTeX, participants were taught how to create augmentations using `\textcolor` to color expressions, `\overbrace` or `\underbrace` to introduce labels with extent markers, and `annotate-equations` [32] to introduce labels with leader lines, including the optional argument `yshift` for adjusting the vertical position of labels.

6.2.4 Tasks. Each participant completed four timed tasks and a single exploratory task. After each task, participants completed a questionnaire reflecting on their experience.

Timed tasks. Participants completed four timed tasks, in two pairs. The first pair of tasks was C1 and C2, which were "creation" tasks. In these tasks, participants created augmentations for an unaugmented formula to match a provided screenshot. Each task required participants to add 3 colors and 3 extent labels.

The second pair of tasks was E1 and E2, which were "editing" tasks. In these tasks, participants were given a formula that was already augmented and asked to modify 4 aspects of the augmentation to match a provided screenshot. This latter pair of tasks was designed to reflect the setting where authors need to interact with augmentation markup when evolving their designs.

Within each pair of tasks, participants completed one task with FFL and one task with the LaTeX baseline. Within pairs, tasks were designed to be as similar to each other in difficulty as possible. Participants were randomly assigned interface and task order within each group of tasks, with the following variations, counterbalancing to reduce the effect of task or interface order:

Task 1		Task 2		Task 3		Task 4	
C1	FFL	C2	LaTeX	E1	FFL	E2	LaTeX
C2	FFL	C1	LaTeX	E2	FFL	E1	LaTeX
C1	LaTeX	C2	FFL	E1	LaTeX	E2	FFL
C2	LaTeX	C1	FFL	E2	LaTeX	E1	FFL

All tasks were timed to compare the speed of completion. A task concluded when a participant completed the task and reported they were done, or when they reached an imposed time limit of 6 minutes and 30 seconds. The facilitator verified completion by comparing the participant's output to a reference result using a rubric that permitted very small differences in color and label position. The task duration was chosen by observing that pilot participants completed most tasks within 5 minutes; we then increased task duration to the longest that could be accommodated in the hour-long study. Over 80% of tasks were completed before reaching the time limit.

Exploratory task. Finally, participants were given 10 minutes to augment a short document resembling the one from Section 3, and asked to augment it in a way that made the formula easier to understand. They were encouraged to explore the augmentation features, and allowed to ask about how to use FFL to achieve their goals. They were also asked to follow the *think-aloud* protocol [40], as demonstrated by their facilitator.

6.2.5 Questionnaire and interview instruments. After each timed task, participants were asked to complete a brief questionnaire reporting how difficult the task was, and to comment on how the interface could have better supported them in their tasks. At the conclusion of the study, participants completed a retrospective questionnaire reflecting on their experience with the interfaces overall. Then, they were interviewed for several minutes as the researcher asked follow-up on questions motivated by observations or responses to the questionnaire.

6.3 Analysis

To examine the effect of interface on task timing and participants' self-reported ease, we fit them with linear mixed-effects models [6]. These models take task, task order, and interface and their interactions as fixed effects, and participant as a random effect. Significance was assessed using an F-test using Satterthwaite's estimate of effective degrees of freedom [64], with p-values corrected by the Holm–Bonferroni method [30]. To compare participants' responses to Likert scale questions about the two interfaces, we performed Wilcoxon signed-rank tests [74]. For these tests, only data from the first 28 of 33 participants was considered, omitting participants who used a personal laptop, and considering a subset for which there was complete balance across interface and task order.

Observation notes, open-ended questionnaire feedback, as well as interview transcripts were analyzed following a thematic analysis approach [5]. Two authors performed an open coding pass, each analyzing half of the observation and questionnaire data and then merging the results. Another two authors reviewed the codes comprehensively. The four authors worked together to revise and organize themes, and to check the alignment between excerpts and themes. One author then reviewed interview transcripts to identify excerpts relating to central themes that emerged from the analysis that had not yet been captured in the observation notes.

7 RESULTS

In this section, we describe our findings. Participants are referred to by pseudonyms *P1–33*. P1–28 were included in our quantitative tests and results. P32–33 completed a variant of the study that involved use of a personal laptop. To convey representativeness of the findings, observations are accompanied with numbers indicating how many participants an observation reflects (e.g., "(5)" means 5 participants).

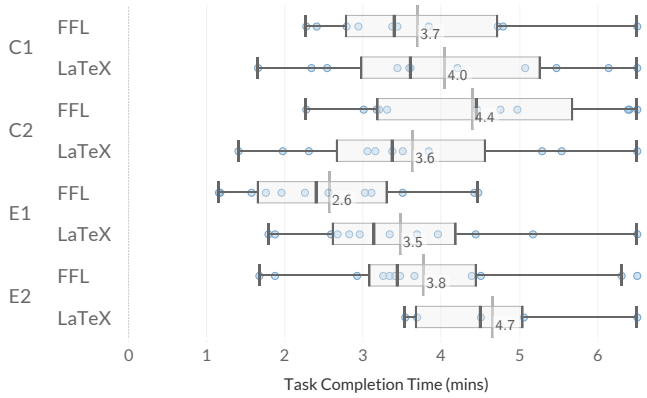


Figure 4: Task completion time. Participants completed tasks E1 & 2 significantly faster with FFL than with LaTeX. Box-and-whiskers depict median, quartiles, and extrema (within 1.5 IQR). An additional, taller vertical line annotates the average. Individual times are rendered as dots in the background. Incompletes are encoded as maximum time. Per-row mean time and standard deviation appear in Table A.2.

7.1 Effect of FFL on task success

Overall, there was significant improvement in task time, self-reported ease, and readability when participants used FFL for editing tasks (E1 & 2), and no perceived difference for creation tasks (C1 & 2).

7.1.1 Completion rate. Overall, participants completed tasks at about the same rate when using FFL and LaTeX. Most participants succeeded in most tasks: altogether, participants reached the time limit on less than 20% of tasks, amounting to 6 failed FFL tasks and 12 failed LaTeX tasks. The most difficult task for LaTeX seemed to be task E2 where 8 participants failed to complete in the LaTeX condition ($p = 0.025$, Fisher's Exact Test [18]). When asked to indicate the extent to which they were able to do what they wanted on a 7-point Likert scale (Figure 5), there was no significant difference between FFL and LaTeX ($F = 0.792$, $p = 0.565$). A complete listing of per-task completion rates appears below in Table A.1.

7.1.2 Speed. As depicted in Figure 4, participants completed the complex editing tasks (E1 & E2) more quickly with FFL than with LaTeX. A linear mixed-effects model found the interface to have a significant effect ($F = 6.7$, $p = 0.02$). Other significant effects include task ($F = 11$, $p = 2 \times 10^{-5}$) and task-interface interaction ($F = 6.8$, $p = 0.001$); task order was not significant. As implied by the task-interface interaction effect, the effect of FFL was stronger for some tasks than others. Fitting the same model to the pairs of creation (C1 & 2) and editing (E1 & 2) tasks separately, the effect of FFL was significant for editing tasks ($F = 27$, $p = 7 \times 10^{-5}$), but not for the creation tasks ($p \approx 1$). While these test statistics are in part

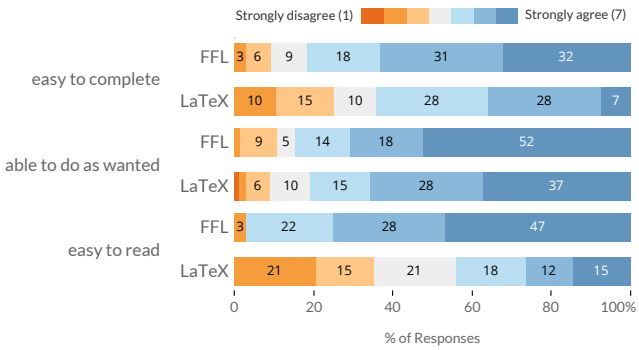


Figure 5: Self-reported ease for timed tasks. On the whole, participants reported greater ease with FFL than with LaTeX. Data comes from responses when participants were asked to indicate agreement (from "strongly disagree" (1) to "strongly agree" (7)) with the statement in the left column of the bar chart. Numbers indicate percentage of total responses relative to the row. Per-row medians and arithmetic means appear in Tables A.3–A.5.

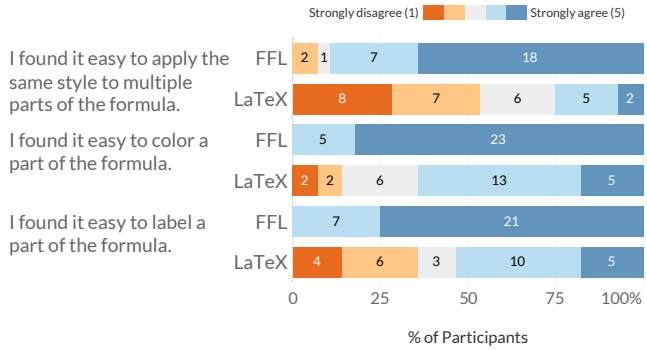


Figure 6: Ease-of-use ratings in retrospective questionnaire. Ease was reported on three dimensions for both FFL and LaTeX (shown in the leftmost labels on the bar chart).

affected by our choice to cut off participants at 6.5 minutes, a visual inspection suggests the trends hold for participants who were not cut off: FFL decreased task time for the 0th–75th quartiles, ranges for which no participants had reached the cutoff limit (see Figure 4). Our observations of participants revealed no clear indications that participants were further from complete when cut off in the FFL condition than in the baseline condition.

7.1.3 Ease. Participants reported significantly higher ease completing tasks with FFL than with LaTeX ($F = 16$, $p = 6 \times 10^{-4}$). On a 7-point Likert scale, participants reported a median score of 7.0, versus 6.0 with LaTeX (Figure 5). Models fit on subsets of tasks showed the difference in ease to be significant for editing tasks E1 & 2 ($F = 19$, $p = 2 \times 10^{-4}$), but not tasks C1 & 2 ($p \approx 1$).

Additional questions on the questionnaire indicate aspects of FFL that might have led to greater ease. Following the editing tasks E1 & 2, participants reported significantly greater ease in reading augmentation code (Figure 5) in FFL than with LaTeX ($F = 23$, $p = 6 \times 10^{-5}$). In the retrospective questionnaire, participants compared the ease of using FFL to LaTeX for a variety of primitive augmentation operations (Figure 6), reporting greater ease with FFL for coloring parts of formulas ($W = 7$, $p < 0.002$, mdn. 5.0 vs. 4.0), labeling parts of formulas ($W = 0$, $p < 0.002$, mdn. 5.0 vs. 4.0), and applying the style to multiple parts of the formulas ($W = 24$, $p < 0.002$, mdn. 5.0 vs. 2.0), on a 1–5 scale.

7.1.4 Differences in success. While on the whole participants reported high levels of comfort with LaTeX in the introductory questionnaire, there was still considerable individual variation in comfort with both LaTeX and CSS. When we fit our model to take background factors into account,[4] we observed years of experience of LaTeX as a significant predictor of task speed ($F = 10$, $p = 5 \times 10^{-5}$), with interface becoming insignificant ($F = 5.4$, $p = .1$). For the creation tasks alone, years of experience with LaTeX is not significant ($p = .3$). For the editing tasks, years of experience is significant ($F = 10$, $p = 3 \times 10^{-4}$), and interface remains a significant effect ($F = 27$, $p = 6 \times 10^{-5}$). Other background factors such as self-reported comfort with LaTeX or CSS were not significant predictors. Overall, additional years of experience of LaTeX reduced task completion times, though the trends vary considerably when broken down by task and interface pair.

7.1.5 Interpretation. In summary, participants completed tasks about as often with FFL and LaTeX. FFL led to quicker completion, with less difficulty. Post-hoc tests showed the effect to be significant for editing tasks E1 & 2, but not creation tasks C1 & 2. We explain this discrepancy with two observations.

First, E1 & 2 were performed after C1 & 2. Some participants reported an initial learning curve with FFL, or encountered gaps or misconceptions regarding FFL during the first pair of tasks. These gaps and misconceptions were sometimes resolved by the time they began the second pair of tasks. Learning effects may provide as a partial explanation: among 33 participants, our observation notes showed 23 participants making 35 critical mistakes (i.e., input that resulted in compilation errors or incorrect outputs) in C1 & 2, reduced to 18 participants making 23 mistakes in E1 & 2.

Second, E1 & 2 required participants to work with considerably more complex and denser markup along with some augmentation already integrated to begin with. E1 & 2 reflect a setting where a formula has been augmented and the authors wish to experiment with alternative designs. We interpret this effect to indicate that FFL manifests more value as augmentation markup becomes larger and the presence of existing augmentation code in the editing tasks served as a scaffold in case of unfamiliarity with FFL's syntax; in the LaTeX baseline, this results in the markup languages becoming increasingly tangled and difficult to evolve, as discussed in greater detail in the next section.

7.2 Effect of FFL on authoring experience

In this section, we review observations, interviews, and questionnaire data to arrive at a comprehensive understanding of how FFL supports, and in some cases works against, the experience of formula augmentation. Overall, participants found FFL's "core" features useful (Figure 7). This section introduces strengths and shortcomings of FFL in terms of the cognitive dimensions of notation [4],

[4]When fitting a model with background factors as fixed effects, we remove the random effect of participant ID.

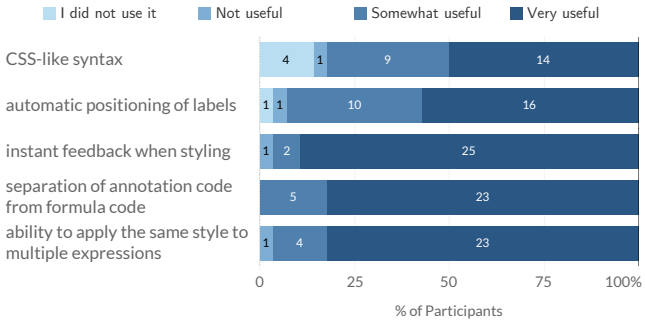


Figure 7: Usefulness of features. Shown are participants reponses to the question "How useful was *[feature]* when you used FFL to augment math formulae?"

a framework used in programming language design to examine and discuss the effect of language design choices.

7.2.1 Strengths. FFL improved the authoring experience as follows:

Viscosity. FFL reduced the number of actions required to accomplish some goals. This was most clear when participants edited augmentations for multiple expressions at once. Participants frequently expressed appreciation for the ability to make cross-cutting changes with a single style specification (5), and wished for a similar capability for LaTeX (4). Making cross-cutting changes was described as more "efficient" (P4) and "easier" (P12, P24) with FFL. All but one participant described the ability to apply one style to multiple expressions as very useful (Figure 7).

Hard mental operations. FFL made it easier for participants to orient themselves to augmentation markup. LaTeX was the less preferred choice for reading markup (Figure 5). LaTeX was described as difficult to read (16) and used complex or unintuitive syntax (6). The association of augmentations with expressions is difficult to understand due to the dependence on copious numbers of nested braces to associate them (14). Reading LaTeX was therefore described as "holding a lot of moving pieces in my mind" (P18), where "it is a nightmare to look for what I am editing" (P13). Reading challenges arose when participants had difficulty identifying expressions to which LaTeX commands applied (2), mapping from parts of the rendered formula to the corresponding LaTeX (5), reading and editing the markup (3), and pinpointing sources of errors (2).

In comparison, FFL seemed easier to read—we rarely heard similar criticisms levied against FFL. 16 participants explicitly mentioned their appreciation for the separation of formula markup from augmentation markup; this division was called a "big advantage" and "very powerful" (P13). The separation of annotation code from formula code was reported as "very useful" by the vast majority of participants in the retrospective questionnaire, and "somewhat useful" by all remaining participants. Participants rated the readability of FFL significantly higher than LaTeX (Section 7.1.3).

Error proneness. FFL removed a class of errors with its approach to associating expressions with augmentations. As mentioned in prior work [27], one challenge of using LaTeX to augment formulas is to use braces correctly to associate augmentations with expressions. Participants described braces as "annoying" (P28), finding it difficult to find matching pairs of braces (6), and desiring the ability to find out which braces are redundant or missing (2). Braces were the most common kind of error we observed: at least some participants made a bracing error for each task (8 participants for task C1; 2 for C2; 6 for E1; and 8 for E2). Participants also encountered issues with using `\def` correctly, writing arguments to commands in the right order, and other LaTeX compilation errors. As noted by participants, FFL did not see these difficulties due to its approach to associating augmentations with expressions (2).

Closeness of mapping. In several situations, FFL provided a close mapping to the ways participants could envision expressing augmentations. Two participants described that the metaphor of CSS, including its use of selectors and attributes, was "intuitive." The design of selectors allowed participants to indicate which expressions they wished to augment by selecting, and then copying and pasting, those expressions from the formula into their FFL specification (2). When asked to indicate the degree to which FFL "did what I expected to," all but 2 participants agreed, and over half of participants strongly agreed.

On the whole, participants developed comfort with a large number of primitives in a short amount of time. By the time they performed the exploratory authoring task, participants had developed enough comfort with the language that they frequently made use of color (24), labels with leader lines (19), and labels with extent markers (11). These augmentations made use of myriad language features, including single character wildcards (15), sequence wildcards (15), unions (11), and the adjustment of label positions (9). See Appendix Section B for examples.

Progressive evaluation. The favorite feature of FFL was the instant feedback supported by the FFL runtime. More participants described this feature as "very useful" than any other feature. 7 participants explicitly indicated their appreciation for instant feedback. In contrast, the LaTeX toolset required slower compilation of the document to see the effect of one's changes to the markup (2), which was described as "not very convenient" (P21).

7.2.2 Shortcomings. While FFL improved the experience of authoring formulas in numerous ways, it also introduced new challenges meriting new solutions to design and training:

Closeness of mapping. FFL was not without a learning curve. Some participants found aspects of the CSS-like syntax challenging (8); this is in part because participants generally had low self-reported comfort with CSS (Section 6.1). Participants also expressed discomfort with the `glob` syntax [37] for wildcards (1), and other aspects of LaTeX's math mode (5). These experiences serve as a reminder that FFL expects familiarity with CSS, glob, and LaTeX. We expect many authors seeking to use FFL in web documents would have this experience; though FFL still imposes a threshold to entry. An additional indicator of a learning curve is that 8 participants reviewed the cheat sheet before beginning their first task with FFL, suggesting that the tutorial was not enough to internalize the syntax. Similar challenges were observed for the LaTeX baseline, with participants forgetting commands taught in the tutorial (3) or failing to properly use commands from the cheat sheet (3).

While participants largely succeeded in selecting expressions with the selector syntax, several participants desired support for direct selection through mouse interaction with the formula (3). Similarly, participants desired the ability to highlight expressions corresponding to a selector (3). Participants also desired code generation (1) and no-code features (2), where style code could be partially or completely generated for the author. Direct selection features are beyond the scope of a language design, though they might serve as useful additions to an editing environment.

Error proneness. FFL removed some classes of errors, though they introduced friction for others. The current runtime provides only very limited support for error tolerance, reporting, and recovery. 6 participants introduced typos and had difficulty understanding why their augmentation markup was not working as intended when they failed to notice those typos. Some of these typos arose from challenges related to "closeness of mapping"—several participants used incorrect delimiters that perhaps best reflected a lack of familiarity with the base CSS syntax. 3 participants wished that FFL continued to render live even when errors were present in the markup. For these reasons, participants desired numerous standard editor affordances that assist in reducing errors, including autocomplete (7), syntax highlighting (2), and templates (3).

Visibility. Any sufficiently complex language contains constructs users are unaware of. We observed several such constructs for FFL that were either undiscoverable, or poorly suited once discovered.

First, participants expressed confusion around scoping augmentations. The default behavior of the FFL runtime is to apply selectors globally across an entire document. What should an author do when they wish for their augmentation rules to apply to only a single expression, formula, or single passage? Several authors had this specific question (8). The current solutions in FFL are: (1) the `intersect` command; (2) an `:nth` selector that selects an indexed occurrence; (3) creating an indexable group in the formula markup by adding brackets around it; (4) using style overriding (i.e., using one rule to style all expressions, and a second rule to revert it for some subset of those expressions). These features were largely unused, perhaps due to issues of discoverability or learnability. At least 2 participants expressed some confusion with overriding.

Second, participants desired more influence over the appearance of labels, including label size (4), color (3), and font weight (3). While the `.ffl-label` class is applied to all labels for just this purpose, participants were not often aware of it. These undiscovered features represent opportunities to either increase visibility, or redesign constructs to be easier to guess.

Expressiveness. Expressiveness is not a cognitive dimension of notation, though we discuss it here as a catchall for controls participants desired that FFL did not provide. One often-desired feature was the ability to assign a single label to multiple expressions simultaneously. For example, in the exploratory task, participants often wanted to create one label for "slope" and connect it via leader lines to all four β terms in the formula (9). The default behavior of FFL is to assign a label to only the first matched expression in a formula. As one participant noted, this made the behavior of FFL inconsistent, because style rules applied to all matching expressions, while labels applied to only the first matched expression (P23). Several participants wished for different behavior from the automatic label layout algorithm (4), and desired the ability to fine-tune label layout beyond FFL's current capabilities (3).

8 DISCUSSION

Our study showed greater speed, ease, and readability of markup code with FFL for the second pair of tasks, which were complex editing tasks. Evidence from the study suggests FFL reduces viscosity, hard mental operations, and error proneness, while providing affordances promoting closeness of mapping and progressive evaluation. These findings suggest the promise of the ideas behind FFL, namely the separation of formula and augmentation markup, live feedback, and its approach to syntax. In this section, we examine the generalizability of the findings, and opportunities for advancing the research agenda of which FFL is a part.

8.1 Limitations

The generalizability of our findings is necessarily limited to tasks and the sample of participants we studied. When interpreting the results, it is useful to take stock of how authors of web-based math documents would differ from participants in the study.

First, we anticipate that authors would have greater motivation to use the tools in naturalistic settings of use. If an author chose to use FFL, it would reflect a desire to make notation more approachable. We expect authors woul experiment more ambitiously with the tools compared to study participants who may not have had any prior experience explaining formulas in their writing.

Second, they would likely be familiar with the formula markup, having written it themselves: for both the LaTeX and FFL conditions, this would likely lead to faster task completion times.

Third, users "in the wild" would not have the luxury of having the tools demonstrated over a 10-minute tutorial, and therefore may have more difficulty in a walk-up-and-use experience.

And finally, their FFL markup would have likely gotten longer if they were augmenting a full-length document. Due to constraints of the in-lab study, we were only able to measure the performance of FFL on mostly single-formula tasks. Although we included cross-cutting changes in all tasks and an open-ended task on a document excerpt as a more realistic scenario, this means that participants in our study did not get a chance to encounter complexities that might arise with longer style specifications and that we did not observe all the difficulties to be seen with scoping augmentation.

Of these limitations, the fourth and fifth are indicators that our lab study reveals only a subset of challenges using FFL; the remaining limitations suggest that task performance could improve for FFL, or both FFL and LaTeX, in more realistic settings. Challenges to using FFL should be further documented by refining the FFL toolkit and evaluating its use in real authoring settings.

8.2 Future work

A first line of future research should address opportunities in extending FFL, some of which revealed in the study (Section 7.2.2).

Scoping. Authors necessarily wish to restrict augmentations to particular expressions, formulas, and passages. While the FFL language provides such capabilities, these were either not discovered or used ineffectively by participants. A future solution could be to

let authors specify local "scopes" of application in the document markup (e.g., labeling individual passages or formulas) in order to refer to them in selectors.

Resilient expression matching. FFL's current approach to matching token sequences leads to some brittleness in matching expressions that are rendered the same way, but have different LaTeX markup (e.g., in the current implementation, `$a_0^1$` matches "`$a_0^1$`" but not "`$a^1_0$`," even though they are rendered identically). This was largely not a problem for participants in the study, though we find this undesirable in our own use. FFL provides some flexibility to address cases like these, but we believe a more robust implementation of FFL may benefit from matching patterns with abstract syntax trees, rather than concrete token sequences.

Further improvements. As noted in Section 7.2.2, FFL should be extended with the ability to apply one label to multiple expressions, more precisely adjust the positions of labels, and better recognize and recover from syntax errors.

This research also points the way to follow-up research on math augmentation that extends into new sorts of tooling.

Direct augmentation. Some participants desired assistance in writing selectors, understanding selections, and expressing styles. They proposed the ability to directly select them, highlight rendered expressions that are matched by selectors, and generate styles (Section 7.2.2). We see FFL as a stepping stone to interactive authoring tools involving direct augmentation like those described by participants, where FFL is used as a substrate, similarly to how backend visualization grammars like Vega-Lite [65] enables visualization exploration interfaces like Voyager [75].

Animated formulas. FFL was designed to augment static texts, like blog articles or online textbooks. What would an augmentation language look like for dynamic presentations of notation, like animations on the popular *3Blue1Brown* [61] YouTube channel for explaining math, where formulas are built up step-by-step and annotated gradually with color and labels? We see FFL as a starting point for developing grammars of animated notation. However, new primitives would have to be designed, as they have in other areas with generalized visualization annotation DSLs for animation [20].

Making texts interactive. One pattern of augmentation is creating interactive formulas, where readers can tinker with the values of expressions and see how it influences downstream computations in the formula [27]. Prior tools like Idyll [11], *Tangle.js* [71], and Potluck [42] envision the creation of parametric documents where values update reactively as users interact with controls. Extensions to FFL could unify such affordances with its syntax, perhaps even taking advantage of the computation a formula represents to automatically map values in one part of a formula to values elsewhere.

Accessibility. Augmentations specified in a language like FFL encode additional meaning about a formula, such as what symbols make up meaningful expressions, and what those expressions mean. This information should ideally be surfaced in a way that is accessible to blind and low-vision readers. FFL could be extended to provide cues to screen readers to read a formula aloud in ways that improve upon the default reading order.

9 CONCLUSION

Our controlled lab study yielded two results. First, in complex editing tasks, FFL led to faster and easier editing of augmentation markup compared to a LaTeX baseline, while yielding more readable markup. Second, for simpler tasks where authors wrote simple augmentations from scratch, we observed no significant differences between the two. Our study offers signs that FFL reduces viscosity, hard mental operations, and error proneness, while supporting closeness of mapping and and progressive evaluation. This paper demonstrates the potential of tools like FFL that extend authoring environments to support the practice of augmenting notation. We hope tools like FFL bring about more pervasive authoring of approachable explanations of math notation.

ACKNOWLEDGMENTS

We thank authors who graciously published articles with augmented formulas which we adapted as demo material and study stimuli [3, 9, 28, 55, 56, Mohammed et al., Murad, Azad, Cockett & Heagy, Hohman et al.], Dr. Rowan Cockett for sharing with us his experience in scientific communication and document editing tools, members and friends of *Penn HCI* for providing early feedback on the tool and the study, as well as open-source contributors who not only created the libraries and tools which we directly cited, but also many indirect dependencies that make what we do possible.

REFERENCES

[1] Thomasenia Lott Adams. 2003. Reading mathematics: More than words can say. *The Reading Teacher* 56, 8 (2003), 786–795.
[2] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. 2005. Evaluation of multimodal input for entering mathematical equations on the computer. In *CHI'05 Extended Abstracts on Human Factors in Computing Systems*. ACM, 1184–1187.
[3] Kalid Azad. https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform. ([n. d.]). Better Explained.
[4] Alan Blackwell and Thomas Green. 2003. Notational Systems—The Cognitive Dimensions of Notations Framework. In *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. Morgan Kaufmann, 103–134.
[5] Ann Blandford, Dominic Furniss, and Stephann Makri. 2016. *Qualitative HCI Research: Going Behind the Scenes*. Morgan & Claypool Publishers.
[6] Violet A Brown. 2021. An introduction to linear mixed-effects modeling in R. *Advances in Methods and Practices in Psychological Science* 4, 1 (2021), 2515245920960351.
[7] Carrie J. Cai and Philip J. Guo. 2019. Software Developers Learning Machine Learning: Motivations, Hurdles, and Desires. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*. 25–34.
[8] David P. Carlisle. 1997. *The* `color` *package*. CTAN-Archive.
[9] Rowan Cockett, Lindsey Heagy, and Doug Oldenburg. 2016. Pixels and their neighbors: Finite volume. *The Leading Edge* 35, 8 (2016), 703–706.
[10] CodeCogs Equation Editor *CodeCogs Equation Editor*. Retrieved July 7, 2023 from https://editor.codecogs.com/
[11] Matthew Conlen and Jeffrey Heer. 2018. Idyll: A Markup Language for Authoring and Publishing Interactive Articles on the Web. In *Proceedings of the Symposium on User Interface Software and Technology*. ACM, 977–989.
[12] Open Source Contributors. markdown-it. https://github.com/markdown-it/markdown-it.
[13] Will Crichton. 2022. A New Medium for Communicating Research on Programming Languages. (2022).
[14] Curvenote *Equation | curvenote.dev*. Retrieved July 12, 2023 from https://curvenote.dev/article/equation
[15] Pierre Dragicevic, Stéphane Huot, and Fanny Chevalier. 2011. Gliimpse: Animating from Markup Code to Rendered Documents and Vice Versa. In *Proceedings of the Symposium on User Interface Software and Technology*. ACM, 257–262.
[16] Anton N. Dragunov and Jonathan L. Herlocker. 2003. Designing intelligent and dynamic interfaces for communicating mathematics. In *Proceedings of the International Conference on Intelligent User Interfaces*.
[17] Emily Eisenberg and Sophie Alpert. *KaTeX*. Retrieved September 16, 2020 from https://katex.org

[18] R. A. Fisher. 1922. On the Interpretation of χ^2 from Contingency Tables, and the Calculation of P. (Jan 1922). https://doi.org/10.2307/2340521

[19] Max Froumentin. *Mathematical Markup Language (MathML).* Retrieved September 16, 2020 from https://www.w3.org/Math/whatIsMathML.html

[20] Tong Ge, Yue Zhao, Bongshin Lee, Donghao Ren, Baoquan Chen, and Yunhai Wang. 2020. Canis: A High-Level Language for Data-Driven Chart Animations. *Computer Graphics Forum* 39 (2020).

[21] Camille Gobert and Michel Beaudouin-Lafon. 2022. i-LaTeX: Manipulating Transitional Representations between LaTeX Code and Generated Documents. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. ACM, 1–16.

[22] Robert L. Goldstone, Tyler Marghetis, Erik Weitnauer, Erin R. Ottmar, and David Landy. 2017. Adapting perception, action, and technology for mathematical reasoning. *Current Directions in Psychological Science* 26, 5 (2017), 434–441.

[23] Google Slides *Google Slides.* Retrieved July 7, 2023 from https://slides.google.com

[24] Han L. Han, Miguel A. Renom, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2020. Textlets: supporting constraints and consistency in text documents. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, 1–13.

[25] Avery Harrison, Hannah Smith, Taylyn Hulse, and Erin R. Ottmar. 2020. Spacing out! Manipulating spatial features in mathematical expressions affects performance. *Journal of Numerical Cognition* 6, 2 (2020), 186–203.

[26] Andrew Head, Kyle Lo, Dongyeop Kang, Raymond Fok, Sam Skjonsberg, Daniel S. Weld, and Marti A. Hearst. 2021. Augmenting Scientific Papers with Just-in-Time, Position-Sensitive Definitions of Terms and Symbols. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*.

[27] Andrew Head, Amber Xie, and Marti A. Hearst. 2022. Math Augmentation: How Authors Enhance the Readability of Formulas Using Novel Visual Design Practices. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. ACM, Article 491, 18 pages. https://doi.org/10.1145/3491102.3501932

[28] Fred Hohman, Andrew Head, Rich Caruana, Robert DeLine, and Steven M. Drucker. 2019. Gamut: A Design Probe to Understand How Data Scientists Understand Machine Learning Models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300809 Paper 579.

[29] Fred Hohman and other contributors. 2020. Awesome Mathematical Notation Design. https://github.com/fredhohman/awesome-mathematical-notation-design

[30] Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70. http://www.jstor.org/stable/4615733

[31] Inkscape *Inkscape.* Retrieved July 7, 2023 from https://inkscape.org/

[32] ST John and other contributors. *Annotate equations in LaTeX using TikZ.* Retrieved July 6, 2023 from https://github.com/st--/annotate-equations

[33] Project Jupyter. jupyterlab-katex. https://github.com/jupyterlab/jupyter-renderers.

[34] Markus Knauff and Jelica Nejasmic. 2014. An efficiency comparison of document preparation systems used in academic research and development. *PloS one* 9, 12 (2014), e115069.

[35] Donald E. Knuth. 1986. *TEX: The Program.* Addison-Wesley.

[36] Labella.js *Labella.js.* Retrieved September 16, 2020 from https://twitter.github.io/labella.js/

[37] Bell Labs. 1971. Unix Programmer's Manual. https://www.bell-labs.com/usr/dmr/www/1stEdman.html. https://www.bell-labs.com/usr/dmr/www/man71.pdf

[38] Joseph J. LaViola and Robert C. Zeleznik. 2006. Mathpad2: a system for the creation and exploration of mathematical sketches. In *ACM SIGGRAPH 2006 Courses*. ACM, 33–es.

[39] Jakob Leitner, Christian Rendl, Florian Perteneder, Adam Gokcezade, Thomas Seifried, Michael Haller, Robert Zeleznik, and Andrew Bragdon. 2010. NiCE Formula Editor. In *ACM SIGGRAPH 2010 Talks*. ACM, 1–1.

[40] Clayton Lewis. 1982. *Using the" thinking-aloud" method in cognitive interface design.* IBM TJ Watson Research Center Yorktown Heights, NY.

[41] Yong Li, Shoaib Kamil, Alec Jacobson, and Yotam Gingold. 2022. HeartDown: Document Processor for Executable Linear Algebra Papers. In *SIGGRAPH Asia 2022 Conference Papers*. 1–8.

[42] Geoffrey Litt, Max Schoening, Paul Shen, and Paul Sonnentag. *Potluck: Dynamic documents as personal software.* Retrieved July 12, 2023 from https://www.inkandswitch.com/potluck/

[43] Digital Science UK Ltd. Overleaf. https://www.overleaf.com/.

[44] Suyu Ma, Chunyang Chen, Hourieh Khalajzadeh, and John Grundy. 2021. Latexify Math: Mathematical Formula Markup Revision to Assist Collaborative Editing in Math Q&A Sites. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–24.

[45] Lars Madsen, Will Robertson, and Joseph Wright. 2014. *The mathtools package.* CTAN-Archive, ctan.org.

[46] Hildebrand et Al. (Open Source Contributors) Majda. 2022. Peggy. https://github.com/peggyjs/peggy. https://peggyjs.org/index.html

[47] Tyler Marghetis, David Landy, and Robert L. Goldstone. 2016. Mastering algebra retrains the visual system to perceive hierarchical structure in equations. *Cognitive research: principles and implications* 1 (2016), 1–10.

[48] Mathcha *Mathcha.* Retrieved July 7, 2023 from https://www.mathcha.io/

[49] David Matthews. 2019. Craft beautiful equations in Word with LaTeX. *Nature* 570, 7760 (2019), 263–264.

[50] Alexandra Mendes, Roland Backhouse, and Joao F. Ferreira. 2014. Structure editing of handwritten mathematics: Improving the computer support for the calculational method. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. ACM, 139–148.

[51] Microsoft. *Write an equation or formula.* Retrieved July 7, 2023 from https://support.microsoft.com/en-au/office/write-an-equation-or-formula-4f799df7-4ca4-4670-afd3-6135768b01d0

[52] Microsoft PowerPoint *Microsoft PowerPoint.* Retrieved July 7, 2023 from https://microsoft.com/powerpoint

[53] Anders Miltner, Sumit Gulwani, Vu Le, Alan Leung, Arjun Radhakrishna, Gustavo Soares, Ashish Tiwari, and Abhishek Udupa. 2019. On the Fly Synthesis of Edit Suggestions. In *Proceedings of the SIGPLAN Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. ACM, 143:1–143:29. Article 143.

[54] Morten Misfeldt. 2011. Computers as medium for mathematical writing. *Semiotica* 2011 (2011), 239.

[55] Haneen Mohammed, Ziyun Wei, Eugene Wu, and Ravi Netravali. 2020. Continuous Prefetch for Interactive Data Applications. arXiv:2007.07858 [cs.DB]

[56] Jousef Murad. 2020. Derivation of the Navier-Stokes Equations. https://www.youtube.com/watch?v=zWdnf3Uh1RE. Jousef Murad | Deep Dive.

[57] Sheshera Mysore, Mahmood Jasim, Haoru Song, Sarah Akbar, Andre Kenneth Chase Randall, and Narges Mahyar. 2023. How Data Scientists Review the Scholarly Literature. In *Proceedings of the 2023 Conference on Human Information Interaction and Retrieval*. ACM, 137–152.

[58] Wode Ni, Joshua Sunshine, Vu Le, Sumit Gulwani, and Titus Barik. 2021. recode: A lightweight find-and-replace interaction in the ide for transforming code by example. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. ACM, 258–269.

[59] Joshua Maxwell Pollock. 2022. Bluefish: A Grammar of Discrete Diagrams.

[60] Patrick Rein, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. 2018. Exploratory and live, programming and coding. *The Art, Science, and Engineering of Programming* 3, 1 (2018), 1–1.

[61] Grant Sanderson. *3Blue1Brown.* Retrieved September 8, 2021 from https://www.youtube.com/c/3blue1brown

[62] Grant Sanderson. 2022. *Manim.* Manim Community. https://3b1b.github.io/manim/

[63] Nazmus Saquib, Rubaiat Habib Kazi, Li-yi Wei, Gloria Mark, and Deb Roy. 2021. Constructing Embodied Algebra by Sketching. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*.

[64] F. E. Satterthwaite. 1946. An Approximate Distribution of Estimates of Variance Components. *Biometrics Bulletin* 2, 6 (1946), 110–114. http://www.jstor.org/stable/3002019

[65] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. In *ACM Transactions of Visualization and Computer Graphics*, Vol. 23. 341–350.

[66] Mary D. Shepherd and Carla C. Van De Sande. 2014. Reading mathematics for understanding—From novice to expert. *The Journal of Mathematical Behavior* 35 (September 2014), 74–86.

[67] Steven L. Tanimoto. 1990. VIVA: A Visual Language for Image Processing. 1, 2 (1990), 127–139.

[68] The LaTeX Project *The LaTeX Project.* Retrieved July 7, 2023 from https://www.latex-project.org

[69] Michael Toomim, Andrew Begel, and Susan L. Graham. 2004. Managing Duplicated Code with Linked Editing. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*. IEEE, 173–180.

[70] Paul Topping. 1999. Using MathType to create TeX and MathML equations. In *Proceedings of the 1999 TEX Annual Meeting, TUGBoat*, Vol. 20.

[71] Bret Victor. *Tangle: a JavaScript library for reactive documents.* Retrieved August 15, 2021 from http://worrydream.com/Tangle/

[72] W3C. *SVG.*

[73] Erik Weitnauer, David Landy, and Erin Ottmar. 2016. Graspable Math: Towards dynamic algebra notations that support learners better than paper. In *2016 Future Technologies Conference (FTC)*. IEEE, 406–414.

[74] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968

[75] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2015. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics* 22, 1 (2015), 649–658.

[76] Katherine Ye, Wode Ni, Max Krieger, Dor Ma'ayan, Jenna Wise, Jonathan Aldrich, Joshua Sunshine, and Keenan Crane. 2020. Penrose: from mathematical notation to beautiful diagrams. *ACM Transactions on Graphics* 39 (2020), 144.

[77] Hsin I. Yung and Fred Paas. 2015. Effects of Computer-Based Visual Representation on Mathematics Learning and Cognitive Load. *J. Educ. Technol. Soc.* 18 (2015), 70–77.
[78] Robert Zeleznik, Andrew Bragdon, Ferdi Adeputra, and Hsu-Sheng Ko. 2010. Hands-On Math: A page-based multi-touch and pen desktop for technical work and problem solving. In *Proceedings of the Symposium on User Interface Software and Technology*. ACM, 17–26.

A DESCRIPTIVE STATISTICS

Below, we show detailed tables and figures of descriptive statistics collected from the usability study.

Task completion

	Task C1	Task C2	Task E1	Task E2
FFL	3	2	0	1
LaTeX	2	1	1	8

Table A.1: Counts of participants who did not complete timed tasks (each count is out of 14 participants).

Time	Task C1		Task C2		Task E1		Task E2	
(s)	*FFL*	LaTeX	*FFL*	LaTeX	*FFL*	LaTeX	*FFL*	LaTeX
$\bar{x}$	253.3	242.8	258.1	229.2	157.3	206.0	223.4	350.4
σ	97.65	101.1	98.6	95.8	66.4	81.3	87.5	68.1

Table A.2: Task completion times, reported as arithmetic means and standard deviations, by task and interface.

Self-reported ease

In the tables below, cells show the mean and median rating across participants on a Likert scale of 1–7, where 1 corresponds to "strongly disagree" and 7 corresponds to "strongly agree" to a statement.

Avg./Mdn. Score (1-7)	Task C1	Task C2	Task E1	Task E2	Exp. Task
FFL	5.47/6.0	5.31/5.5	6.38/6.5	5.44/5.5	6.30/6.0
LaTeX	5.00/5.0	5.24/5.0	5.13/6.0	3.61/3.5	N/A

Table A.3: Participants' self-reported ease by task and interface. Participants were asked to indicate their agreement with the statement "It was easy to complete the task."

Avg./Mdn. Score (1-7)	Task C1	Task C2	Task E1	Task E2	Exp. Task
FFL	6.06/7.0	5.69/6.5	6.63/7.0	5.44/5.5	6.30/7.0
LaTeX	6.00/6.0	6.29/7.0	5.86/6.0	4.72/5.0	N/A

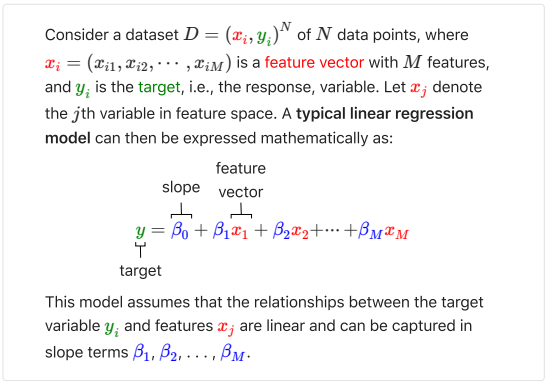
Table A.4: Participant self-reported efficacy by task and interface. Participants were asked to indicate their agreement with the statement "I was able to do what I wanted with the tool."

Avg./Mdn. Score (1-7)	Task E1	Task E2
FFL	6.25/6.0	6.00/6.5
LaTeX	5.06/5.0	3.61/3.0

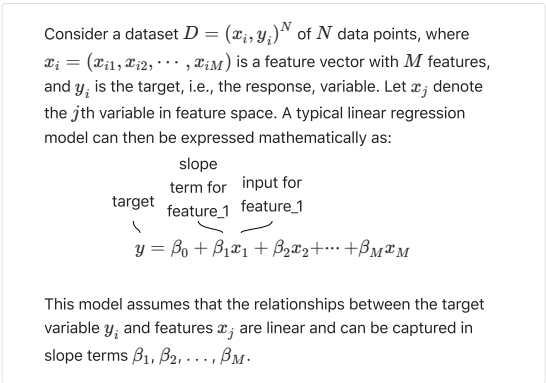
Table A.5: Participant self-reported sense of readability. Participants were asked to indicate their agreement with the statement "I found it easy to read the styling code/specification."

B EXAMPLE AUGMENTATIONS

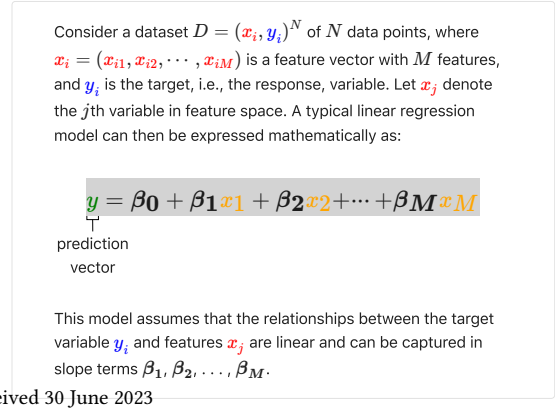
Below, we show examples of augmentations authors performed in the open-ended authoring task on again [28, Hohman et al.]. The following passage from P26 is representative of most participants' finished work. It makes use of color to relate expressions to descriptions in the text, and labels to explain several expressions.



Other participants took different approaches. For instance, P13 used labels alone, believing them to be sufficient for a textbook-style passage (and that color was better suited for personal notes):



Some participants experimented more ambitiously with CSS, when they had sufficient prior knowledge. For instance, P17 experimented with `background-color`, `font-size`, and `font-weight` and expressions, in addition to the other typical augmentations.



Received 30 June 2023