Typst: A Modern Typesetting Engine for Science

Andrey Voynov[1], Alberto Corbi[2*], Pau López-Oliver[3], David Gil[4]

[1] Bauman Moscow State Technical University (BMSTU), 2nd Baumanskaya, 5, Moscow (Russia)
[2] Universidad Internacional de La Rioja (UNIR), Av. de la Paz, 137, Logroño, La Rioja (Spain)
[3] Universitat de València (UV), Carrer del Dr. Moliner, 50, Burjassot, Valencia (Spain)
[4] Agencia Estatal de Meteorología (AEMET), Leonardo Prieto Castro, 8, Madrid (Spain)

* Corresponding author: alberto.corbi@unir.net

Received 14 October 2025 | Accepted 2 February 2026 | Early Access 19 February 2026

Abstract

Typst is a modern, markup-based document preparation system designed for producing professional documents with a strong emphasis on simplicity, performance, and powerful scripting for structured content and automated layout and styling. This paper presents a comprehensive review of the platform and its surrounding ecosystem through the systematic and comparative analysis, highlighting key features, some unique to this approach, that are relevant to both general-purpose and niche use cases. It introduces the system and compares its output and functionality with LaTeX, outlines the historical development of document technologies and the core layout and composition algorithms they rely on, and explains the language through visual examples accompanied by corresponding source code. The paper also examines the official Typst compiler and web application, as well as adoption trends and ecosystem growth. It concludes with cross-domain case studies in Physics, Math, and Computer Science, intended for beginners and readers interested in modern document workflows and systems.

Keywords

Incremental Parsers, Markup Languages, Paper Composition, Performance, Reproducibility, Typesetting Systems.

DOI: 10.9781/ijimai.2026.2269

I. Introduction

Typst is a new markup-based typesetting language (and its tooling ecosystem) for technical and scientific documents. It is designed to be an alternative to both advanced environments like LaTeX and simpler tools like Microsoft Word and Google Docs. The goal with Typst is to build a typesetting framework that is highly capable, extensible, reliable, fast and very easy to use. For instance, with Typst, it is possible to:

- Create professional-quality documents with ease.
- Access extensive functionality, including mathematical typesetting, figure management and an auto-generated table of contents.
- Utilize powerful templates that automatically apply consistent formatting during the writing process (easy content reuse).
- Benefit from high-quality typographical output with uncompromising word- and character-level justification and overall layout.
- View changes instantly with real-time preview functionality.
- Make use of clear and understandable error messages.
- Apply a consistent styling to fonts, margins, headings, lists, etc.
- Collaborate seamlessly with other authors.

According to the arXiv preprint repository, submissions have steadily grown, peaking at over 24,226 in October 2024, underscoring strong demand in academia for tools like Typst to produce high-quality papers[1].

The main contributions of the present paper include: systematic analysis, comparative evaluation, and cross-domain case studies.

The Typst realm comprises a refined and easy-to-understand markup language for defining the content, structure and style of a document, a reasonably fast (and community-driven) document renderer, and a companion web application that enables real-time in-browser compi-lation. All these components will be explored in Section IV, Section V, and Section VI, respectively.

Additionally, the project hosts a repository of extensions (packages and templates) called Typst Universe, discussed in Section VIII. However, given that Typst is becoming popular, and its adoption is growing steadily over the years (as observed in Section VII), it is now often compared against the well-established LaTeX system. For this reason, we will start with a quick analogy of both environments (Section II).

For the sake of completeness, Section IX, Section X, and Section XI will focus on the application of this new typesetting system in more specific science and engineering domains such as Theoretical Physics, Cosmology, Chemistry, Mathematics, Algorithmics, Signal Processing, Computer Science and the composition of slides in technical realms (Section XII). Finally, some conclusions are presented in Section XIII.

[1] https://info.arxiv.org/about/reports/2024_arXiv_annual_report.pdf

Please cite this article as:
A. Voynov, A. Corbi, P. López-Oliver, D. Gil. Typst: A Modern Typesetting Engine for Science, *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 9, no. 7, pp. 107-120, 2026, https://doi.org/10.9781/ijimai.2026.2269

II. Typst and LaTeX

Typst and LaTeX (and variants like LuaTeX or XeTeX) are both markup-based typesetting systems (whose foundations are analyzed in Section III), but they differ in several key aspects [1], [2]. Regarding the language and its syntax, Typst employs intuitive patterns, similar to those found in Markdown [3], making it more accessible. Its commands and language rules are designed to work consistently, reducing the need to learn different conventions for each new add-on (called *packages* in the Typst *semantic field*, and reviewed later in Section V.E).

Fig. 1 shows a side-by-side example of the equivalent LaTeX and Typst code. As it can be seen, even for a small document describing simple, introductory Algebra-related concepts, any LaTeX distribution would need a more cumbersome syntax and auxiliary markup.

Typst

```
= Affine Spaces
Let $A$ be a linear subspace of another space $V$. Then, $A$ is a Euclidean and affine linear space if there exists a mapping: $ f: A times A -> V $ where $A times A$ means _linear mapping that takes a tuple from $A$_.
#let bo(var) = $bold(upright(var))$
= Example
Let the mapping $f$ take two elements $bo(a) = (a_x, a_y)$ and $bo(b) = (b_x, b_y)$ from $A subset.eq RR^2$ and do the following with them: $ f(bo(a), bo(b)) = (a_x - b_x, a_y - b_y) $
```

LaTeX

```
\documentclass{article}
\usepackage{amssymb}
\begin{document}
\section*{Affine Spaces}
Let $A$ be a linear subspace of another space $V$. Then, $A$ is a Euclidean and affine linear space if there exists a mapping: \[ f: A \times A \rightarrow V \] where $A \times A$ means \emph{linear mapping that takes a tuple from $A$}.
\section*{Example}
Let the mapping $f$ take two elements $\mathbf{a} = (a_x, a_y)$ and $\mathbf{b} = (b_x, b_y)$ from $A \subseteq \mathbb{R}^2$ and do the following with them:
\[ f(\mathbf{a}, \mathbf{b}) = (a_x - b_x, a_y - b_y) \]
\end{document}
```

Fig. 1. Typst vs. LaTeX comparison example.

The output of the Typst part in Fig. 1 is rendered in Fig. 2 (although the LaTeX one would be very similar with additional styling like from Fig. 7).

Affine Spaces

Let A be a linear subspace of another space V. Then, A is a Euclidean and affine linear space if there exists a mapping:

$$f : A \times A \to V$$

where $A \times A$ means *linear mapping that takes a tuple from A*.

Example

Let the mapping f take two elements $\mathbf{a} = (a_x, a_y)$ and $\mathbf{b} = (b_x, b_y)$ from $A \subseteq \mathbb{R}^2$ and do the following with them:

$$f(\mathbf{a}, \mathbf{b}) = (a_x - b_x, a_y - b_y)$$

Fig. 2. Output from the Typst code in Fig. 1 and styling from Fig. 7.

Focusing on the renderer and local installs, Typst offers very fast (milliseconds) and incremental compilations, which allows for document previews that are delivered almost right away (for the average human perception). These rendering operations often occur under the so-called Doherty threshold, i.e., below this point, users stay highly productive. However, above it (around 400 milliseconds, as evinced by [4]), system delays quickly degrade user performance and satisfaction.

The compiler (tackled in Section V) is a single lightweight binary (less than 50 MiB) that, when necessary, downloads external packages on-demand, ensuring minimal and secure installations. All operations take place in user space (no need for special administrator privileges).

Regarding the operating procedure, unlike LaTeX, Typst does not require boilerplate code to start a new document: just by creating an empty text file with a typ extension suffices. To simplify further, the Typst project hosts its own online editing service (Section VI). Currently, in the LaTeX world, this can only be achieved through external cloud solutions, such as Overleaf [5]. A very short summary on the main differences between the two ecosystems is presented in Table I.

TABLE I. Main Differences Between LaTeX and Typst

Feature	LaTeX	typst
Syntax	Command-based (`\command{arg}`)	Markdown-inspired (`= Heading`, `_italic_`) + code mode (`#func()`)
Math Mode	`$...$` or `\[...\]`, verbose (`\frac{}{}`)	`$...$`, concise (auto-fractions, variants, etc.)
Headings	`\section{}`, `\subsection{}`	`= Heading`, `== Subheading`
Lists	`itemize`/`enumerate` environments	`-` (bullets), `+` (numbers), `/ Term:` (descriptions)
Commands	Macros (`\newcommand`)	1st-class functions (`#let f(x) = x + 1`, composable
Compiler	Slow (s) and multi-pass	Fast (ms) and incremental
Packages	Large TeX distributions	Cached downloads
Errors	Cryptic	User-friendly, detailed
Graphics	TikZ, PSTricks, etc.	CeTZ, Lilaq, Fletcher, etc.
Team work	Overleaf (third-party)	Own app (Section VI)
Code blocks	Very package-dependent (Listings, Minted, etc.)	Own native support for code blocks (`#raw(code)`)
Citations	Managed externally	Built-in (also Hayagriva)
Deploy	Can be initially heavy (~5 GiB) for most distros	Starts with a single binary (~45 MiB)

III. State of the Art

Markup languages and typesetting systems offer several key advantages, including the separation of content from presentation, which allows authors to focus on structure and semantics while ensuring consistent formatting. They also enable automation, such as dynamic content generation, cross-referencing, and bibliography management, reducing human error and improving efficiency.

A. Typesetting Systems and Markup Languages

Modern typesetting relies heavily on computers, with most printed materials now created digitally rather than through traditional methods like typewriters or movable type. Professional desktop publishing tools offer precise control over elements such as kerning and ligatures, and more general-purpose tools like LibreOffice, Microsoft Word, Google Docs, and WPS Office have adopted some of these features [6]. Still, these general applications lack the full suite of comprehensive typesetting capabilities, such as high-quality hyphenation or the ability to flow text across multiple custom regions. This notable gap has driven adoption of text-based systems, especially in academia, where LaTeX dominates due to its powerful formula rendering and flexible layout control.

These setups rely on compiling source text into formatted output like PDF, separating content from presentation to allow the easy reuse and adaptation of document styles [7]. Typesetting systems are designed not only to produce high-quality visual documents, but also to support the complex process of creating structured content. A well-designed system must consider numerous layout features such as *line* and *page breaking*, *kerning*, *ligatures*, contextual *glyph positioning*, and the treatment of languages with varied directionalities.

Additionally, avoiding formatting issues like *widows* (the last line of a paragraph stranded at the top of a page) and *orphans* (the first line of a paragraph left alone at the bottom of a page) is part of achieving professional-quality results. However, this visual precision is only one side of the coin. These systems must also support complex content like sections, tables, and figures in a structured manner (Table II).

TABLE II. Most Important Typesetting Algorithms

Challenge	Description	Algorithm/Approach
Paragraph breaking	Breaking text into lines with aesthetically pleasing spacing/ hyphenation	Knuth-Plass line breaking algorithm [8]
Justification	Lines align evenly at margins	Spacing adjustments with the Knuth-Plass [9]
Grid Layout	Optimal space for rows/ columns in grids	Constraint-based layout calculation [10]
Page Breaking	Page division while respecting layout	Greedy + backtracking algorithms [11]
Glyph Selection	Correct glyphs depending on context	Font shaping and con-text-sensitive glyphs [12]
Bidirectional Text	Mixing LTR and RTL scripts	Unicode Bidirectional Algorithm [13]
Incremental Layout	Reusing layout computations after small edits	Constraint-based layout cache/region reuse [14]
Styling	Consistent styles	Programmable layouts
Unicode	Modern scripts, ligatures, and grapheme clusters	Shaping and grapheme line breaking [15]

Historically, the development of markup-oriented systems began in the 1960s with tools like Runoff, and evolved significantly with programs like Troff [16] and TeX. Troff brought enhanced typographic features to Unix environments, while TeX revolutionized typesetting with its advanced paragraph layout algorithms and extensible macro system. LaTeX, built on top of TeX, pushed the concept further by introducing *descriptive markup*, where authors focus on the logical structure of content rather than its appearance. Parallel to this, systems like GML, SGML, and eventually HTML and XML developed the idea of defining structure through custom tags [17], with SGML forming the basis for later web standards. Over time (Fig. 3), styling systems like CSS and XSL emerged to handle the transformation of structured content into presentational formats [18]. Yet, limitations persisted, such as verbosity in XML and complexity in LaTeX customization.

Recent efforts in the typesetting world have aimed at modernizing older systems. Lightweight languages like Markdown[2] or AsciiDoc[3] prioritize ease of use but sacrifice power. For this reason, these tools usually team up with conversion solutions, such as Pandoc [19].

On the other hand, advanced software like LuaTeX [20] or ConTeXt[4] attempt to replace TeX while maintaining its output quality. However, these often inherit TeX's core limitations, like performance or syntax issues. LaTeX has slowly evolved with modular improvements like the L3 layer and a new hook system [21]. Nevertheless, many challenges remained unsolved around usability, accessibility, and automation.

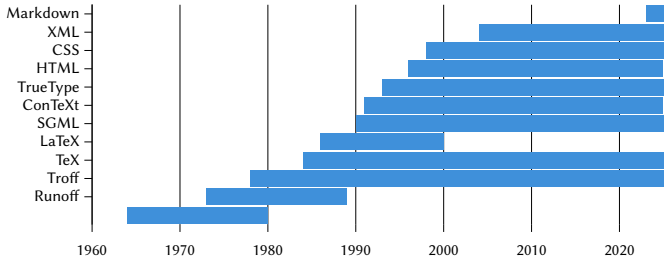

Fig. 3. Evolution of some Typesetting technologies.

B. Computed Documents and Dynamic Content

Dynamic content generation is a crucial feature of modern markup languages and typesetting systems, enabling documents to update automatically based on external data or user input. By integrating pro-gramming logic, such as loops, conditionals, and variables, these systems can produce data-driven outputs, from dynamically generated reports in LaTeX to interactive web pages in Markdown with embedded scripts. This capability reduces manual repetition, minimizes errors, and ensures consistency when dealing with large or evolving datasets. Furthermore, dynamic generation supports real-time updates in interactive documents, such as dashboards or educational materials, enhancing usability and engagement. By blending structured markup with computational power, these systems bridge the gap between static documents and flexible, automated publishing workflows, making them indispensable for tech-nical, scientific, and web-based documentation.

Knitr [22], Sweave [23], and similar computational document applications, such as RMarkdown [24] and Jupyter Notebook[5], integrate code execution with document authoring, allowing authors to embed live code chunks that produce figures, tables, and statistical results within a narrative. These systems are particularly prevalent in data science and scientific writing, where reproducibility is crucial. Built on top of LaTeX or Markdown, they provide a powerful, albeit often complex, workflow that couples typesetting with dynamic content generation.

In contrast, Typst offers a more unified and modern approach: rather than embedding a separate scripting language into markup, it merges typesetting and computation into a single, consistent language. This seamless integration allows Typst to support sophisticated layout logic, styling, and even data-driven approaches without the verbosity or complexity found in the aforementioned tools. Besides, when teaming up with modern web technologies such as WebAssembly (or Wasm, discussed in Section V.F), the possibilities are almost endless. For instance, the package Pyrunner[6] allows the execution of arbitrary chunks of Python code within a Typst document (Fig. 4).

```
#import "@preview/pyrunner:0.3.0" as py                    Typst
#let compiled = py.compile(```py
import re
def find_emails(string):
    return re.findall(r"\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.
[a-zA-Z]{2,}\b", string)
def sum_all(*numbers):
    return sum(numbers)
```)
#let text = "My email address is john.smith@example.com and my
friend's email address is jane.doe@example.net."
Result: #py.call(compiled, "find_emails", text)
#py.call(compiled, "sum_all", 1, 2, 3)
```

Result: ("john.smith@example.com", "jane.doe@example.net") 6

Fig. 4. Python code and its output produced by Pyrunner.

[2] https://daringfireball.net/projects/markdown

[3] https://asciidoc.org

[4] http://wiki.contextgarden.net

[5] https://jupyter.org

[6] https://typst.app/universe/package/pyrunner
```

Other current WebAssembly-grounded integration solutions for com-putational documents in Typst are:

Neoplot[7] for generating plots with Gnuplot (Fig. 17).

Jlyfish[8] for integrating Julia code.

Callisto[9] for reading and rendering Jupyter notebooks.

Diagraph[10] for binding simple Graphviz-based diagrams (Fig. 5).

Nulite[11] for plotting Vega-based charts.

Jogs[12] a native JavaScript runtime.

```
#import "@preview/diagraph:0.3.6" : raw-render
#raw-render(```dot
digraph G {
  rankdir=TD
  LaTeX -> typst
  LaTeX[color=gray, fontname="New Computer Modern", fontsize=18, fontcolor=lightblue]
  typst[color=DodgerBlue, fontname="Buenard", fontsize=18, fontcolor=royalblue]
}
```)
```

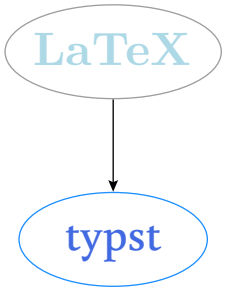


Fig. 5. Example of a Graphviz diagram, rendered natively with Wasm.

## IV. The Markup Language

Typst employs straightforward markup syntax for standard formatting operations. For instance, headings can be created with the = symbol, while text can be italicized by enclosing it in _underscores_.

Typst employs three distinct syntactical modes: markup, math, and code. By default, a .typ document operates in *markup mode*, which handles standard text formatting. *Math mode* enables the composition of mathematical expressions, while *code mode* provides access to Typst's scripting capabilities for dynamic content generation. Transitions be-tween these modes are governed by specific markers (Table III).

TABLE III. Typst Syntactical Modes

| Mode | Syntax | Example |
|---|---|---|
| Code | Prefix code with # | `Number: #(1 + 2)` |
| Math | Surround math with $...$ | `$-x$ is the opposite of $x$` |
| Markup | Put markup in [...] | `#let name = [*Typst!*]` |

All this content is written in Unicode [25]. Typst has embraced this computing standard as a first-class citizen (Fig. 6), making it much more modern and intuitive than traditional typesetting systems.

### A. Styling

Typst makes styling documents flexible, and consistent with a modern and declarative approach. In certain ways, it operates very similarly to CSS, but built for typesetting. With more detail, it uses a consistent and hierarchical styling model where styles can be defined globally, applied to specific elements, or inherited through logical relationships. Typst leverages a declarative syntax combined

```
$ Σ_(α = 1)^γ α = β(β+1) / 2 $
${(x, 💩), (x, 🔥), (y, 💩)} \
  A times B = {(🧃, 🍵) | 🧃 in B}$
#table(
  columns: 5,
  gutter: 2pt,
  stroke: 0.5pt,
  [炬], [باتك ], [🍵], [🤡], [Δ],
  [#emoji.checkmark], $=>$, emoji.alien,
  sym.angstrom, [👻],
)
```

Typst

$$\sum_{\alpha=1}^{\gamma} \alpha = \frac{\beta(\beta+1)}{2}$$

$$\{(x, 💩), (x, 🔥), (y, 💩)\}$$

$$A \times B = \{(🧃, 🍵) \mid 🧃 \in B\}$$

炬	باتك	🍵	🤡	Δ
✓	⇒	👽	Å	👻

Fig. 6. Illustration of Unicode use in Typst for text and math.

with programmatic features, allowing users to define reusable styles, functions, and templates. For example, document-wide settings like fonts, margins, and colors can be set at the beginning, while local overrides can be applied to headings, tables, or other elements using rules and CSS-inspired selectors. This ensures clean separation of concerns between content and presentation with flexibility. Additionally, Typst's styling system supports dynamic adjustments, such as conditionally changing layouts or automating typographic refinements, thanks to its built-in scripting capabilities.

Typst uses a two-pronged styling system to separate styling from content. First, `set` rules allow the declaration of global or local scoped defaults for elements (like general text, pages, paragraphs, headings, figures, alignments, lists, etc.), specifying parameters such as font, size, margins, justification, line spacing, numbering, and layout. Once set, these defaults apply automatically wherever that element appears. Second, `show` rules enable custom rendering logic for specific elements[13] by providing customizable CSS-like selectors[14].

Normally, `show` and `set` statements are combined (`show-set` included) to tweak an element's appearance through various code-based trans-formations (e.g., small caps, run-in headings, added logotypes, text color and fonts, table background, etc.). They can also reliably theme an entire document (templates) via an *everything* `show` rule. By combining traditional typesetting with modern programming, Typst provides a powerful and intuitive way to effectively manage document styling for academic papers, technical reports, or dynamic publications. For instance, the rules necessary to produce Fig. 2 are presented in Fig. 7.

Global set rule

```
#set text(9pt)
```

Show and show-set rules for heading styling

```
#show heading: set text(9pt)
#show heading: set align(center)
#show heading: it => block(smallcaps(it.body))
```

Typst

Fig. 7. Example of a global `set` rule and `show-set` & `show` rules (on the `heading` element) necessary to render the content of Fig. 2.

B. Control Structures

Typst incorporates several control structures that facilitate dynamic content generation and conditional logic within documents. The `if` statement enables conditional rendering, allowing content to be included or excluded based on specific conditions. In combination with `set`, `set page(fill: rgb("#333333")) if` dark applies a dark theme when the dark variable is `true`. It is important to note that `set` rules are scoped, thus, applying them *within* an `if` block confines their effect to that block's scope. Additionally, Typst treats `if` as an expression, permitting concise inline conditionals like `if x > 10 {"High"} else {"Low"}`.

[7] https://typst.app/universe/package/neoplot

[8] https://typst.app/universe/package/jlyfish

[9] https://typst.app/universe/package/callisto

[10] https://typst.app/universe/package/diagraph

[11] https://typst.app/universe/package/nulite

[12] https://typst.app/universe/package/jogs

[13] https://typst.app/docs/reference/foundations/function/#element-functions

[14] https://typst.app/docs/reference/foundations/selector
```

For iterative operations, Typst offers `for` and `while` loops. The `for` loop is versatile, capable of iterating over strings, arrays, dictionaries, and more. For example, `for letter in "abc" {letter}` processes each character in the string separately. Control statements like `break` and `continue` are available to manage loop execution, allowing for early exits or skipping iterations. The `while` loop continues execution as long as the specified condition remains true. These loops can be utilized within content blocks to dynamically generate document elements, such as populating tables or lists based on data structures. Finally, arrays and dictionaries can be iterated with object oriented-like methods present in modern programming languages (`.map`, `.filter`, `.enumerate`, etc.).

C. Math

Typst offers robust support for mathematical expressions, providing a syntax that is both intuitive and powerful. To enter math mode, it is only necessary to enclose mathematical expressions within dollar signs (`$ ... $`). For display-style equations, spaces or newlines can be added between the dollar signs and the content. Typst's math mode supports a wide range of symbols and functions, including Greek letters (Φ, Ω, Λ, etc.), operators (dim, ker, Pr, etc.), and more (ℑ, 𝕂, ⋀, 𝔐, etc.). Subscripts and superscripts are handled using the underscore (_) and caret (^) symbols, respectively. For example, `$x^2$` renders as x^2, and `$a_b$` renders as a_b. Additionally, Typst automatically scales delimeters like parentheses and brackets to fit their content, similar to LaTeX's `\left` and `\right` commands. This ensures that complex expressions are rendered accurately. For instance, the math code in Fig. 8 allows the typesetting of the Lagrangian of the Standard Model of Particle Physics.

```
$                                                                    Typst
  cal(L) = -1 / 4 B_(mu nu) B^(mu nu) - 1 / 8 tr(mathbf(W)_(mu nu)
  mathbf(W)^(mu nu)) - 1 / 2 tr(bold(G)_(mu nu) G^(mu nu)) \
  + (macron(nu)_L, macron(e)_L) tilde(sigma)^mu i D_mu vec(nu_L,
  e_L) + macron(e)_R sigma^mu i D_mu e_R + macron(nu)_R sigma^mu i
  D_mu nu_R + "h.c."\
  - sqrt(2) / mu [macron(nu)_L, macron(e)_L] phi M^e e_R +
  macron(e)_R macron(M)^e macron(phi) vec(nu_L, e_L) \
  - sqrt(2) / mu [(- macron(e)_L, macron(nu)_L) phi^* M^nu nu_R +
  macron(nu)_R macron(M)^nu phi^T vec(-e_L, nu_L)] \
  + (macron(u)_L, macron(d)_L) tilde(sigma)^mu i D_mu (u_L, d_L) +
  macron(u)_R σ^mu i D_mu u_R + macron(d)_R σ^mu i D_mu d_R +
  "h.c.   " \
  - sqrt(2) / mu [(macron(u)_L, macron(d)_L) phi M^d d_R +
  macron(d)_R macron(M)^d macron(phi) vec(u_L, d_L)] \
  - sqrt(2) / mu [(- macron(d)_L, macron(u)_L) phi^* M^u u_R +
  macron(u)_R macron(M)^u phi^tack.b vec(-d_L, u_L)] \
  + (D_mu phi)^dagger D^mu phi - m_h^2 [macron(phi) phi - nu^2 /
  2]^2 / (2 nu^2)
$
```

Fig. 8. Typst code for the Lagrangian of the Standard Model.

$$
\begin{aligned}
&\mathcal{L} = -\frac{1}{4}B_{\mu\nu}B^{\mu\nu} - \frac{1}{8}\,\mathrm{tr}\left(\mathbf{W}_{\mu\nu}\mathbf{W}^{\mu\nu}\right) - \frac{1}{2}\,\mathrm{tr}\left(\boldsymbol{G}_{\mu\nu}G^{\mu\nu}\right)\\
&+(\bar{\nu}_L, \bar{e}_L)\tilde{\sigma}^{\mu} i D_{\mu}\begin{pmatrix}\nu_L\\ e_L\end{pmatrix} + \bar{e}_R\sigma^{\mu} i D_{\mu} e_R + \bar{\nu}_R\sigma^{\mu} i D_{\mu}\nu_R + \mathrm{h.c.}\\
&-\frac{\sqrt{2}}{\mu}[\bar{\nu}_L, \bar{e}_L]\varphi M^{e} e_R + \bar{e}_R\bar{M}^{e}\bar{\varphi}\begin{pmatrix}\nu_L\\ e_L\end{pmatrix}\\
&-\frac{\sqrt{2}}{\mu}\left[(-\bar{e}_L, \bar{\nu}_L)\varphi^{*} M^{\nu}\nu_R + \bar{\nu}_R\bar{M}^{\nu}\varphi^{T}\begin{pmatrix}-e_L\\ \nu_L\end{pmatrix}\right]\\
&+(\bar{u}_L, \bar{d}_L)\tilde{\sigma}^{\mu} i D_{\mu}(u_L, d_L) + \bar{u}_R\sigma^{\mu} i D_{\mu} u_R + \bar{d}_R\sigma^{\mu} i D_{\mu} d_R + \mathrm{h.c.}\\
&-\frac{\sqrt{2}}{\mu}\left[(\bar{u}_L, \bar{d}_L)\varphi M^{d} d_R + \bar{d}_R\bar{M}^{d}\bar{\varphi}\begin{pmatrix}u_L\\ d_L\end{pmatrix}\right]\\
&-\frac{\sqrt{2}}{\mu}\left[(-\bar{d}_L, \bar{u}_L)\varphi^{*} M^{u} u_R + \bar{u}_R\bar{M}^{u}\varphi^{\top}\begin{pmatrix}-d_L\\ u_L\end{pmatrix}\right]\\
&+\left(D_{\mu}\varphi\right)^{\dagger} D^{\mu}\varphi - m_h^2\frac{\left[\bar{\varphi}\varphi - \frac{\nu^2}{2}\right]^2}{2\nu^2}
\end{aligned}
$$

Fig. 9. Rendering of the code in Fig. 8.

This theory, describing three of the four known fundamental forces (electromagnetic, weak and strong interactions), is rendered in Fig. 9.

Beyond basic syntax, Typst allows for advanced customization of mathematical expressions. Matrices can be defined using the mat function, which accepts semicolon-separated rows and comma-separated columns, such as `$mat(1, 2; 3, 4)$` to render a 2×2 matrix. Typst also supports piecewise functions through the cases function, enabling the definition of functions with multiple conditions in a clear format. Moreover, text can be incorporated within math expressions by enclosing it in double quotes, like `$x > 0 "if" y < 1$`. For users who prefer using Unicode symbols directly, Typst accommodates this as well, allowing for a more natural input of mathematical notation.

Besides, the Typst Universe (Section VIII) site hosts a variety of math-related packages to enhance mathematical typesetting:

Quick-Maths package allows users to define custom shorthands for complex expressions, streamlining the writing process.

Great-Theorems provides structured environments for theorems, lemmas, and proofs with customizable styling and numbering.

Game-Theoryst facilitates the typesetting of payoff matrices.

Physica offers tools for scientific and engineering mathematics, including matrix operations and vector calculus (Fig. 10).

Equate enhances the formatting and numbering of mathematical equations, improving readability and reference.

MiTeX integrates LaTeX math syntax into Typst, allowing users to write equations using familiar LaTeX commands.

D. Drawing Capabilities

Typst's visualize module[15] offers a comprehensive suite of tools for creating vector graphics and data visualizations directly within documents. It supports a variety of shapes and elements, including circles, ellipses, rectangles, squares, lines, polygons, and Bézier curves, each customizable with parameters like fill, stroke, and radius.

```
#import "@preview/physica:0.9.8": *
$
  curl (grad f), tensor(T, -mu, +nu),
  pdv(f, x, y, [1,2]) \
  H(f) = hmat(f; x, y; delim: "[",
  big: #true) \
  arrow.l quad arrow.long quad
  chevron.l.double quad ✠ \
  arrow.r quad arrow.l.long.squiggly
$
The Ehrenfest theorem is:
$
  d/(d t) expval(A) = 1/(i hbar)
  expval([A, H]) + expval(delta A/
  (delta t))
$
```

$$\nabla \times (\nabla f), T_{\mu}{}^{\nu}, \frac{\partial^3 f}{\partial x \partial y^2}$$

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

$$\leftarrow \quad \longrightarrow \quad \langle\!\langle \quad ✠$$

$$\rightarrow \quad \rightsquigarrow$$

The Ehrenfest theorem is:

$$\frac{d}{dt}\langle A\rangle = \frac{1}{i\hbar}\langle [A, H]\rangle + \left\langle \delta\frac{A}{\delta t}\right\rangle$$

Fig. 10. Example of advanced math with the Physica package.

The module also allows for the inclusion of images (both raster and vector) and shapes (lines, circles, polygons, etc.), and supports advanced styling options such as gradients (Fig. 11) and tiled patterns (Fig. 12).

It is worth mentioning the Typst-based CeTZ library. CeTZ is a graphics package designed for the Typst typesetting system, aiming to provide capabilities similar to those of LaTeX's TikZ for creating vector graphics [26]. While TikZ is a mature and powerful tool within the LaTeX ecosystem, known for its extensive features, CeTZ is tailored to integrate seamlessly with Typst's syntax and design philosophy.

[15] https://typst.app/docs/reference/visualize

```
#stack(                                    Typst
  spacing: 2pt,
  rect(fill:
  gradient.linear(..color.map.viridis),
  width: 2.5cm, height: 1.8cm),
  rect(fill:
  gradient.radial(..color.map.rainbow),
  width: 2.5cm, height: 1.8cm),
)
```

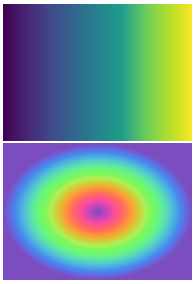

Fig. 11. Gradient stack showing Typst drawing capabilities.

```
#import "@preview/board-n-pieces:0.6.0" : *
#board(
  fen("r4rk1/pp2Bpbp/1qp3p1/8/2BP2b1/
  Q1n2N2/P4PPP/3RK2R" ),
  square-size: 11pt,
  black-square-fill: rgb("#0098cd"),
  white-square-fill: white,
)
```

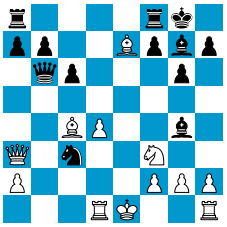

Fig. 12. Chessboard tiled pattern (with the Board-n-Pieces package).

Regarding data visualization, Typst also offers powerful capabilities through its extensible package ecosystem, enabling users to create high-quality plots and charts directly within their documents. Two prominent packages facilitating this are Lilaq and CeTZ-Plot. The first one provides a user-friendly interface for scientific data visualization, drawing inspi-ration from tools like Matplotlib [27] and the TikZ-based PGFplots package (Fig. 13). It emphasizes ease of use, allowing for quick creation of plots with minimal code, and supports features like customizable color cycles, axis configurations, and various plot types.

```
#import "@preview/lilaq:0.5.0" as lq
#let (x, y) = lq.load-txt(read("data.csv"))
#lq.diagram(
  width: 100%,
  height: 25mm,
  lq.plot(x, y),
)
```

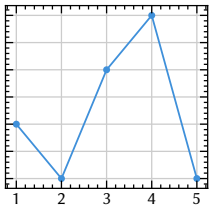

Fig. 13. Example of a plot made with the Lilaq plotting package.

On the other hand, CeTZ-Plot extends the CeTZ drawing library, offering functionalities for creating plots and charts within the CeTZ canvas environment (Fig. 14). It supports various chart types, including pie charts, bar charts, cycles, breadcrumb patterns, and pyramids.

```
#import "@preview/cetz-plot:0.1.3" : *
#import "@preview/cetz:0.4.2"
#set text(weight: "bold")
#let colors = (
  lime, green, olive
).map(c => c.lighten(30%))
#cetz.canvas(smartart.cycle.basic(
  ([Reduce], [Reuse], [Recycle]),
  step-style: colors,
  radius: 1.4,
))
```

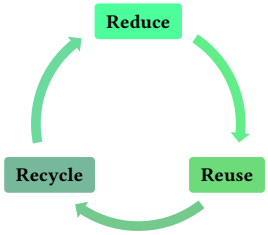

Fig. 14. Example of a cycle diagram created with CeTZ-Plot.

E. Bibliographic References

Typst offers integrated support for bibliographic references. It allows authors to include citations in their documents using the `cite` function or the `@key` notation. The `@` syntax allows referencing not only entries from bibliography files, but also figures, headings, equations, and footnotes.

Currently, both BibLaTeX `.bib` files [29] and Hayagriva `.yaml` files are supported as sources for bibliographic data (Fig. 15). The system utilizes the Citation Style Language (CSL) to format citations and bibliographies [30], providing a wide range of built-in styles such as APA, MLA, IEEE, and Chicago. Furthermore, users can also add custom CSL files to accommodate specific formatting requirements, enhancing overall flexibility in citation management.

```
@article{Corbi23,                          BibTeX
  title     = {Cloud-
  Operated Open Literate
  Educational Resources:
  The Case of the
  MyBinder},
  author    = {Corbi,
  Alberto and Burgos,
  Daniel and Pérez,
  Antonio María},
  journal   = {IEEE
  Transactions on Learning
  Technologies},
  volume    = {17},
  pages     = {893--902},
  year      = {2023},
}
```

```
Corbi23:                                   YAML
  type: article
  title: "Cloud-Operated Open
  Literate Educational Resources:
  The Case of the MyBinder "
  author:
    - Corbi, Alberto
    - Burgos, Daniel
    - Pérez, Antonio María
  date: 2023
  page-range: 893-902
  parent:
    type: periodical
    title: IEEE Transactions on
    Learning Technologies
    volume: 17
```

Fig. 15. Sample of a BibLaTeX entry and its Hayagriva equivalent for [28].

Hayagriva[16] is a Rust-based bibliography management library devel-oped side-by-side with Typst to work smoothly with it. Hayagriva introduces a YAML-backed format for bibliographic entries and incor-porates a CSL processor to format both in-text citations and reference lists. This alternative to BibLaTeX supports all styles provided in the official CSL repository, offering users access to over 2,600 citation styles.

V. The Compiler and Command-Line Interface

Typst's compiler operates differently from traditional ones by using a reactive model (Fig. 16) that tracks dependencies and selectively re-evaluates only the modified parts of a document [31]. This enables instant previews during editing, as the system interprets layout instructions, styling rules, and content in real time [32]. A consistent styling system and an intelligent layout engine work together to resolve these elements efficiently, supporting complex features like math typesetting, dynamic templates, and figures while maintaining responsiveness.

A. Typst Abstract Syntax Tree and Rust

The compilation itself follows a structured yet flexible process. First, the input text is parsed into an *abstract syntax tree* (AST) using Typst's grammar rules, followed by static analysis to resolve imports, variables, and functions. After type checking and evaluating expressions, the AST is transformed into an *intermediate representation* containing layout directives. The compiler then computes the final document layout using a constraint-based algorithm to determine positioning, sizing, and breaks (such as pages). Finally, it renders the output based on the resolved layout. This incremental approach ensures that updates are processed efficiently, minimizing recomputation when changes occur.

Typst's choice of Rust [33] as its underlying programming language provides several key benefits, including high performance, memory safety, and modern tooling. Rust's efficiency allows Typst to compile documents significantly faster than traditional LaTeX systems, with benchmarks showing near-instantaneous updates after initial compi-lation (e.g., 200 ms for changes in a 77-page document). The language's

[16] https://github.com/typst/hayagriva

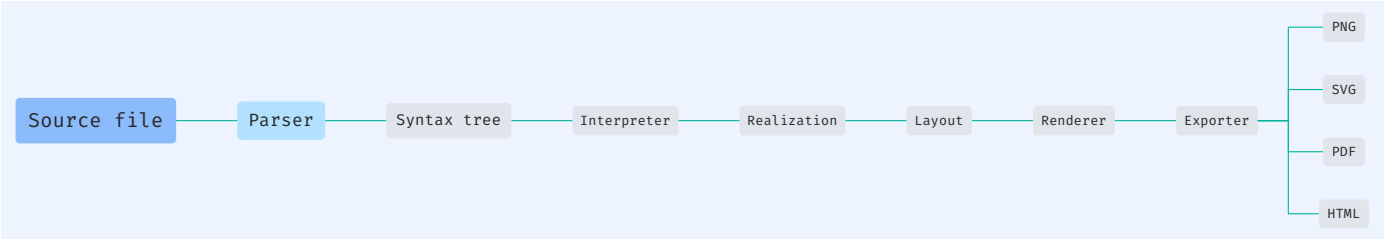

Fig. 16. Typst compiling process, comprising four main phases: *parsing*, *evaluation*, *layout*, and *rendering*. HTML export is in an experimental state.

memory safety guarantees prevent common bugs like use after free and race conditions, which is critical for a typesetting system handling complex document structures. Additionally, Rust's strong type system and zero-cost abstractions enable Typst to implement features like cross-platform development, including Wasm for browser-based tools.

B. Abstraction and Mutation

Abstractions in Typst enables users to manage complexity by hiding irrelevant details through two primary mechanisms:

Functions are defined using `let` syntax and support required positional arguments, optional named arguments with defaults, and *argument sinks* for variadic inputs. A key convenience is the shorthand for passing content as the last argument(s) via brackets (`[ ... ]`). Functions implicitly join multiple values (e.g., `content`) and return them without requiring an explicit return statement.

Modules can be imported (`#import`) for later use or included (`#include`) for immediate inlining of their content.

C. Value Semantics and Coercion

Typst employs *value semantics*, meaning values are treated as if they are copied whenever they are passed. This approach prevents unintended side effects and simplifies reasoning about code. For instance, modifying a dictionary inside a function or during iteration does not affect the original structure because the function or loop receives a copy. This avoids common pitfalls such as cyclic structures, iterator invalidation, and unintended global mutations. As a result, code becomes easier to test/debug, and features like multi-threading are safer/simpler to imple-ment. In Typst, even function arguments and global variables behave as immutable within the scope of a function, reinforcing this isolation.

Although value semantics suggest potential performance costs due to frequent data copying, Typst mitigates this concern with a *copy-on-write* strategy. This means that data is only duplicated when it is modified, allowing it to be efficiently shared across various references without creating unnecessary overhead. This offers a practical balance between performance and clarity. Unlike some languages that explicitly distin-guish between *owned* and *shared* data, Typst keeps its model implicit, which aligns well with its role as a typesetting tool. Users can focus on layout and content creation without needing to manage complex memory models. For example, mutating an array inside a function does not affect the original array, ensuring straightforward data manipulation.

A unique feature is *implicit coercion*: values like numbers or strings are automatically converted to content when used in markup (as well as all other data types). For instance, `#(1 + 2)` in markup becomes 3, while 3.0 retains its fractional part in arrays for debugging clarity. Strings differ from content: even though they can be implicitly or explicitly coerced to content, special syntax for smart quotes, references, etc. will not be picked up, e.g., `#"'Hello'"` will produce 'Hello' and not ‘Hello’ (`eval`, introduced in Section XI.C, can help with such issues).

D. Modules

As mentioned, Typst's other form of abstraction is modules. There are three ways to import a module:

`#import "mymodule.typ"` makes mymodule *accessible*. Then, it is possible to write `mymodule.functionality` to access what is defined with `#let` functionality `= ...` in `mymodule.typ`.

`#import "mymodule.typ": *` puts functionality directly into scope, so that the prefix is not needed anymore.

`#include "mymodule.typ"` inlines the *content* (and only the content) from `mymodule.typ`.

Modules do not give the user/developer any way to mark items as *public* or *private*. Thus, every let statement in the whole module is exported (public). Although this can be solved with re-exports.

E. Packages

As with LaTeX and as commented above, Typst also supports the ad-dition of functionalities via packages. A Typst package is a self-contained collection of Typst source files and assets, structured around a mandatory `typst.toml` manifest file located at the package root (written in the *Tom's Obvious Minimal Language*[17]). This manifest specifies essential metadata such as the package's `name`, version, and `entrypoint`, which points to the main `.typ` file to be evaluated upon import[18]. Additional optional keys like `authors`, `license`, and `description` can also be included. The internal organization of the package is flexible, allowing authors to structure files and directories as they see fit, provided that the `entrypoint` path is correctly specified. All paths within the package are resolved relative to the package root, ensuring encapsulation and preventing access to files outside the package.

Packages are typically stored in a directory hierarchy following the pattern `{namespace}/{name}/{version}` and can be imported into Typst documents using the syntax `#import "@{namespace}/{name}:{version}"`. For local development or experimentation purposes, packages can be placed in designated local data directories, making them accessible without the need for publishing to the shared repository. The @preview namespace in Typst serves as a dedicated space for various community-contributed packages. These packages are hosted in the Typst Universe (backed, in turn, by a GitHub repository).

F. Web Technologies

As introduced in Section III.B, Typst leverages Wasm to enable its core functionalities to run efficiently in web environments [34]. This approach allows Typst to execute its typesetting engine directly within web browsers, facilitating seamless integration into web-based applica-tions and services. By compiling its Rust-based codebase to Wasm, Typst ensures consistent performance across different platforms without the need for native installations. This strategy not only enhances accessibility but also simplifies the deployment process, making Typst a versatile tool for developers and content creators alike.

[17] https://toml.io

[18] https://github.com/typst/packages/blob/main/docs/manifest.md

As an example, the Neoplot package is a specialized tool designed to integrate Gnuplot (a powerful open-source plotting engine [35]) into Typst documents (Fig. 17).

```
#import "@preview/neoplot:0.0.3" as gp
#gp.exec(```gnuplot
set terminal svg font "Fira Math, 34"
size 700,930
set size ratio 2
set border 0
set xzeroaxis linetype 0 linewidth 2
set xtics axis
set xrange [-3:3]
set yrange [-1:9]
set yzeroaxis linetype 0 linewidth 2
set margins 5,5,0,0
plot x**2 lw 6 lc 3 title 'x^2' ```)
```

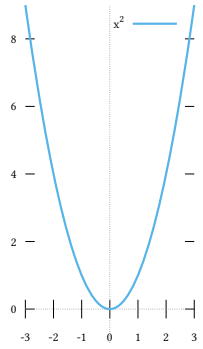

Fig. 17. Parabola plot with the Neoplot Wasm-based package.

The Grayness package allows the application of complex image manipulation algorithms such as blur and contrast (Fig. 18).

Fig. 18. Complex image manipulation via the Grayness Wasm plugin.

G. Security

The Typst compiler ensures safety by implementing strict security measures that prevent potentially harmful operations during document compilation. It restricts file access to the project's root directory, disal-lowing reading or writing files outside this scope, thereby safeguarding against unauthorized data access. In other words, it runs in a sandboxed environment that prevents arbitrary code execution and limits access to the underlying system. This means features like *shell escape* from the TeX world is prohibited [36], [37]. Networking capabilities are restricted to downloading Typst packages and new compiler versions from trusted websites. These design choices collectively create a secure environment, making Typst safe to use even with untrusted input.

H. Introspection

Typst's introspection system provides a suite of functions that enable dynamic interaction with a document's structure and content. Central to this system are functions like `counter` and `query`. The `counter` function allows for tracking and manipulating counts of elements such as pages, headings, figures, and equations. Users can access current counter values, display them in various formats, and even define custom counters for specific needs. For instance, it is possible to create a custom counter to number specific elements uniquely throughout the document. On the other hand, the `query` function facilitates searching the document for elements that match certain criteria, such as all headings of a specific level or elements with certain labels. This is particularly useful for generating dynamic content like table of figures, as it allows for real-time retrieval and display of relevant elements based on the document's current state. Introspection functions require an explicit or implicit `context` to work.

Complementing these are functions like `here`, `locate`, and `metadata`, which offer deeper insights into the document's structure:

`here` retrieves the current location within the document, which can be used together with other functions to determine positional information. For example, combining `here` with `query` can yield the number of specific elements preceding the current point.

`locate` identifies the position of a specific element, allowing for precise referencing or manipulation based on location.

`metadata` enables embedding arbitrary values without producing visible content, which can later be retrieved using `query`. This is particularly useful for storing and accessing auxiliary infor-mation that informs document behavior or content generation.

I. Integrated Development Environments

Typst integrates seamlessly with existing integrated development environments (IDE), such as VSCodium[19] (Fig. 19). For instance, the Tinymist[20] extension provides a comprehensive toolkit for Typst document creation. It offers features such as syntax highlighting, real-time preview, code completion, and error diagnostics, enhancing the editing experience. Users can initialize Typst projects using built-in templates, make use of LSP-enhanced formatting, and manage local packages directly within the editor. Similar feature set is available in Zed, Neovim, Helix, and Emacs. These tools collectively transform almost any text editor or an IDE into a powerful solution for Typst-based typesetting.

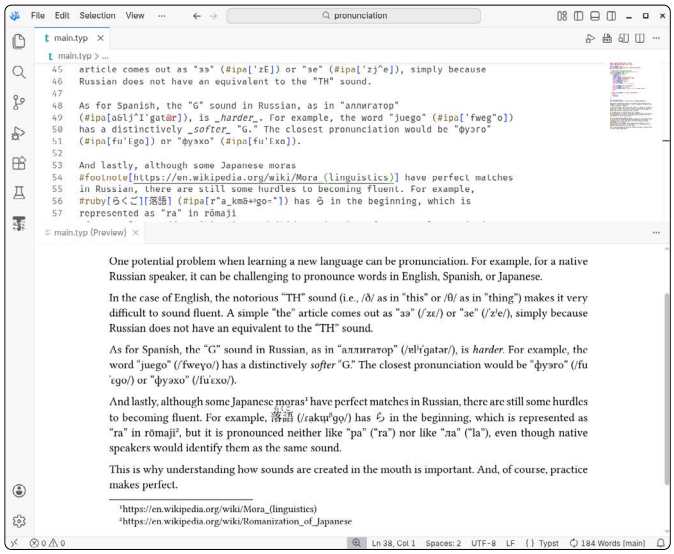

Fig. 19. Creation of a document in VSCodium with Tinymist extension (the links in the figure are interactive via the `place` and `link` functions).

J. Export Options

As of Q4 2025, Typst exports four formats: PDF, SVG, PNG, and HTML. PDF export is the most mature, producing high-quality, resolution-independent documents compliant with PDF 1.7 (also supporting 1.4, 1.5, 1.6, and 2.0). Tagged PDF is enabled by default; all PDF/A archival standards and PDF/UA-1 for enhanced accessibility are supported. Page ranges and standards can be specified via the command-line interface or the web app. SVG export is well-supported

[19] https://vscodium.com

[20] https://github.com/Myriad-Dreamin/tinymist

for embedding vector graphics into web pages, allowing per-page SVG files with customizable naming and page-range selection. PNG export offers the same options as SVG but as raster images. HTML export is experimental (Fig. 16), under active development, and requires a dedicated feature flag enabled.

K. Image Formats

Supporting many image formats is convenient but increases compiler size because each format requires handling code. Typst's maintainers limited supported formats to the most popular: PNG, JPEG, GIF (static-only), and SVG. Since version 0.14.0, WebP is added, PDF images can be embedded as-is, giving a small file-size increase, sharp vector graphics, and selectable text. The feature is already available in the web application.

L. Limitations

Typst has notable limitations. Compilation time and RAM scale with document size and complexity: builds can take tens of seconds or more, and consume 2–7 GiB. Contributing factors may comprise aggressive caching, Wasm plugins, computational code, and complex styling (justification, references, floats, etc.). Missing features include: EPUB and multi-page HTML export, multiple bibliographies (Alexandria, Pergamon), variable fonts, interactive forms, runaround (Meander), vertical writing. Advanced state/context styling has a steep learning curve, and institutional academic adoption, although healthily growing around Master and PhD theses, is still somewhat low (see Section VII).

VI. Web Application

The shift to cloud is transforming content creation and academic work. Platforms like Binder [28], Overleaf, Google Docs, and Notion[21] show that cloud services lower barriers, enable collaboration, and enhance workflows. Education and research are now more dynamic and efficient.

Papeeria and Authorea offer similar collaborative LaTeX editing capabilities but have smaller user bases. PLMLatex, developed by the National Centre for Scientific Research (CNRS), is a French-language LaTeX editor based on the open-source version of Overleaf. It provides a user interface and functionality closely resembling Overleaf, though it lacks certain premium features. CoCalc also supports LaTeX editing alongside tools for calculations, research, and collaboration.

The Typst online editor (the web app, Fig. 20) is a collaborative, web-based platform designed for creating and typesetting documents with Typst. It offers a seamless writing experience with features like instant preview, syntax highlighting, and autocompletion, making it ideal for composing academic papers, technical reports, and other long-form documents. The editor splits the interface into two panels: a source panel for writing Typst markup and a preview panel that renders the document in real time. Users can easily format text, insert images (for this, drag and drop gesture can be used), equations, and bibliographies, and leverage Typst's scripting capabilities for advanced customization. The web app also supports collaboration through the WebSocket standard [38], allowing users to share projects, track changes, and integrate with tools like Zotero and Mendeley for reference management.

The development team is actively working on improvements, includ-ing better mobile usability and additional features like offline PWA support and private templates for teams. The editor is available for free with basic features, while a *Pro* subscription[22] unlocks advanced aspects like Git integration, presentation mode, and increased storage.

[21] https://www.notion.com

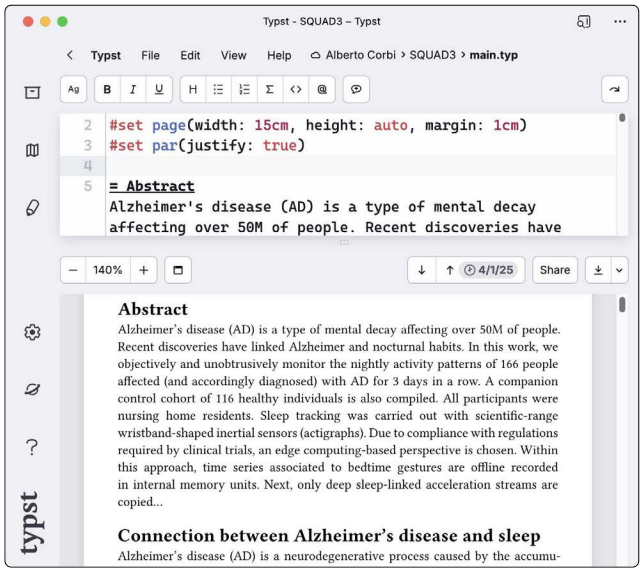

Fig. 20. Screenshot of the typst.app web application (online editor).

Because of Typst's high compilation speeds and instant preview, the web app can even support interactive games, such as a fully functional Tetris. It is published as the Soviet-Matrix package.

VII. Adoption of Typst

Typst has attracted significant interest since its public beta launch and the open-sourcing of its compiler in March 2023. The platform's user-friendly syntax and modern features have attracted a flourishing community, with its GitHub repository amassing over 50,000 stars and~2,700 members in the official online forum. Typst is used by members of over 3,500 universities and laboratories and over 1,000 businesses[22]. Typst's open-source nature and active development suggest a promising future as it continues to evolve and address the needs of its users.

During the period 2020–2025, Typst evolved from a niche LaTeX alternative into a widely adopted document-formatting tool. Early development (2020–2022) focused on core features like a Rust-based compiler, attracting tech-savvy users. By 2023, public beta releases and improved documentation spurred initial growth, though gaps like CJK support persisted. In 2024, corporate adoption (e.g., in banking software) and features like CeTZ for graphics expanded its reach. Projections for 2026 hinge on addressing accessibility and localization, while compiler optimizations (e.g., faster builds) and community tools (e.g., Tinymist, commented in Section V.I) aim to solidify its position[23]. The Typst community is also providing templates for the well-known journals, as evinced in Fig. 21 for the IEEE organization and the MDPI editorial.

Finally, although not its intention, the online service typst.app can also be used as a scientific preprint dissemination platform. Scientific preprint repositories like arXiv[24] and HAL[25] play a crucial role in the rapid publication of research findings across various academic disciplines [39]. These platforms allow researchers to share their work before peer review, enabling quick access to new ideas. The number of Typst-based articles on arXiv is growing, with ~100 research preprints accumulated so far (after analyzing the Computer Science category from 2023 to 2025).

[22] Businesses such as: Dotphoton, Zerodha, IABG, Neodyme.

[23] https://github.com/qjcg/awesome-typst

[24] https://arxiv.org

[25] https://hal.science

Fig. 21. Some Typst-based journal templates already qualified to be used for editorial purposes: Institute of Electrical and Electronics Engineers[26], and Multidisciplinary Digital Publishing Institute[27].

VIII. Typst Universe

Typst Universe[28] is an online platform that offers a curated collection of over 1,000 templates and packages designed to automate Typst documents. Users can find resources ranging from thesis templates to visualization tools, all aimed at simplifying the document creation process. The platform allows users to search, browse categories, and submit their own contributions, fostering a collaborative environment. Some of the packages present on this site are briefly described in Table IV.

TABLE IV. Some Examples of Packages in Typst Universe

Package	Description
Touying	A powerful package for creating presentation slides.
Unify	Simplifies the typesetting of numbers, units, and ranges, similar to LaTeX's Siunitx package [40].
Finite	Renders finite automata diagrams using CeTZ.
Tiaoma	A barcode generator that supports various barcode types by compiling Zint[29] to Wasm.
Problemst	Template for problem sets, homework, or assignments.
Quill	Quill is a package for quantum circuit diagrams.

IX. Application of Typst for Theoretical Physics

Typst's robustness, powerful features and intuitive syntax make it an all-in-one tool to create texts with publication-quality figures. For instance, Penrose-Carter diagrams[30] (PCd) are a way of sketching the entire spacetime of a given spacetime manifold in general relativity on a single, finite sheet of paper. By applying a *conformal* transformation (one that preserves angles but adjusts distances), these diagrams bring infinity to a finite boundary while preserving the light cone structure, so that the global causal layout is immediately visible. PCd simplifies the understanding of black holes, cosmological models, and other relativistic effects. In Typst, it is possible to create a PCd using the CeTZ package. For instance, the PCd associated with the Kruskal extension of the Schwarzschild spacetime is displayed in Fig. 22. CeTZ combines precise mathematical typesetting with programmatic control over geometry.

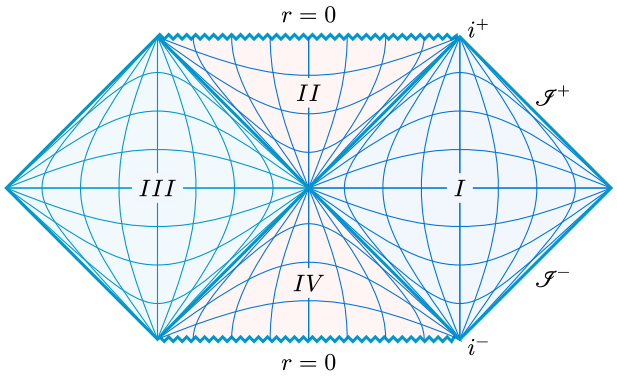

Fig. 22. Penrose-Carter diagram of the Schwarzschild manifold.

In addition to spacetime visualizations, Typst's CeTZ package can be applied in Particle Physics through the creation of Feynman diagrams. Physicists relate the initial and final states of a physical system via the scattering matrix, or S-matrix [41]. The S-matrix is a complex object that has to be perturbatively calculated as a sum of infinite terms. Feynman diagrams are pictorial representations of these terms, each depicting one of the potentially infinite interaction processes that lead to the same final state. A Feynman diagram for the $e^+e^- \to e^+e^-$ scattering process at one-loop order in QED is depicted in Fig. 23.

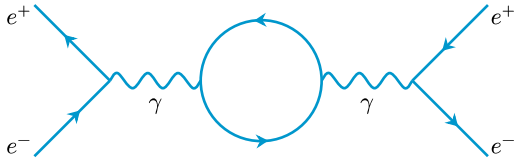

Fig. 23. A Feynman diagram for the $e^+e^- \to e^+e^-$ at one-loop order. The e^+ and e^- annihilate, producing a photon (γ). This photon then becomes a virtual electron-positron pair, which subsequently produces another photon. Finally, the photon becomes the scattered e^- and e^+.

Another common diagram type in Math and present in some branches of Theoretical Physics, is the commutative one. For example, in [42], when speaking of classical field theory on fiber bundles, the commutative diagram shown in Fig. 24 appears, and we can reproduce it using the Commute package, a library designed to draw such diagrams.

$$\begin{array}{ccccccccccc} \cdots & \longrightarrow & \mathcal{O}^{n-1}_\infty & \xrightarrow{d} & \mathcal{O}^{n}_\infty & \xrightarrow{d} & \mathcal{O}^{n+1}_\infty & \xrightarrow{d} & \mathcal{O}^{n+2}_\infty & \longrightarrow & \cdots \\ & & \downarrow{\scriptstyle h_0} & & \downarrow{\scriptstyle h_0} & & \downarrow{\scriptstyle \rho} & & \downarrow{\scriptstyle h_0} & & \\ \cdots & \longrightarrow & \mathcal{O}^{0,n-1}_\infty & \xrightarrow{d_H} & \mathcal{O}^{0,n}_\infty & \xrightarrow{\delta} & \boldsymbol{E}_1 & \xrightarrow{\delta} & \boldsymbol{E}_2 & \longrightarrow & \cdots \end{array}$$

Fig. 24. Cochain morphism of the de Rham complex of the differential graded algebra $\mathcal{O}^*_\infty$ of all exterior forms on finite order jet manifolds (modulo pull-back identification) to its variational complex.

The flexibility of the CeTZ package enables the creation of a wide range of diagrams, while many other packages specialize in convenience and ease of use. Moreover, the near real-time output preview, intuitive syntax and possibility of collaboration enable Typst to be used as a tool to develop complex concepts around Physics, Cosmology, and Mathematics, not just communicate them via papers, books, etc.

[26] https://ieee.org

[27] https://mdpi.com

[28] http://typst.app/universe

[29] https://zint.org.uk

[30] https://en.wikipedia.org/wiki/Penrose_diagram

X. More on Mathematics and Scientific Notation

A. Annotated Mathematics

The Mannot package stands out as a didactic enhancement for math-ematical documents. It enables authors to label and annotate individual parts of equations (Fig. 25), offering an effective means to clarify and explain their components step by step.

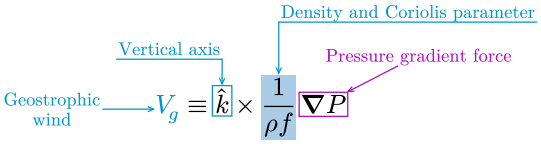

Fig. 25. Mannot-annotated math expression.

Using Mannot, authors can insert visual callouts alongside concise textual explanations that are aligned with terms or sub-expressions within a given formula. This approach transforms dense mathematical notation into explanatory material that is well-suited for textbooks, lectures, or tutorials. This can be taken even further when the same concept or equation is marked in the same color throughout the text. That way, the reader can connect the different parts and concepts much faster.

The didactic layout can be fine-tuned with fully customizable annota-tions. The result is a document that not only presents mathematical content but also actively facilitates learning and comprehension.

B. Physics and Chemistry

Scientific typesetting can be cumbersome, but packages like the aforementioned Physica (Fig. 10) make it straightforward. Physica provides concise, compact, and semantically meaningful commands for advanced mathematical notation, ranging from linear spaces/ algebra to tensor and quantum-mechanical expressions. For vector calculus, `grad`, `curl`, and `div` or `laplacian` can be used: $\nabla \times \mathrm{f}$, $\div \vec{v}$, $\nabla\varphi$, $\nabla^2 u$. With specific commands for differentials and derivatives, first-order, mixed partials, and higher orders are automatically formatted. For instance, the code `$dd(x), dv(T,t), pdv(P,x), pdv(rho,y, 2)$` renders as:

$$\mathrm{d}x, \frac{\mathrm{d}T}{\mathrm{d}t}, \frac{\partial P}{\partial x}, \frac{\partial^2 \rho}{\partial y^2}.$$

Using tensor and quantum notations is also an effortless task with Physica. For instance: `$tensor(h,+mu,+nu)$` will be presented as $h^{\mu\nu}$, and `$bra(u)$` will be shown as $\langle u|$. There is even a way to visualize digital signals with convenient built-in procedures (Fig. 26).

Fig. 26. Signals rendered with the Physica package.

Nuclear and chemical reactions can be typeset with Physica and Typsium (similar to the Mhchem LaTeX package), respectively:

- $^{211}_{83}\mathrm{Bi} \rightarrow {}^{207}_{81}\mathrm{Tl} + {}^{4}_{2}\mathrm{He}$ (Physica),
- $[\mathrm{Co(H_2O)_6}]^{2+} + 4\mathrm{Cl}^- \rightleftharpoons [\mathrm{CoCl_4}]^{2-} + 6\mathrm{H_2O}$.

Finally, Atomic allows the drawing of electronic shells (Fig. 27).

```
#import "@preview/atomic:1.0.0": atom
#let atom = atom.with(
  color: rgb("#eaf6fd"),
  orbitals: 0.7,
  center: 0.4,
  step: 0.34)
#atom(8, 16, "O", (2, 6))
```

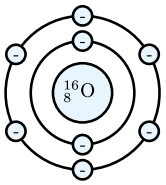

Fig. 27. Atom shells rendered with the Atomic package.

XI. Application of Typst for Computer Science

Computer Science (CS) is a diverse field that covers algorithms and information theory, as well as computer hardware and software. The Typst ecosystem can already accommodate it with many general-purpose packages and even more niche ones.

A. Algorithms & Hardware

For example, to visualize algorithms, the Algorithmic package can be used for creating pseudocode syntax.

Also, the Matofletcher package (an abstraction over the Fletcher one) turns out very useful for creating flowcharts, as shown in Fig. 28.

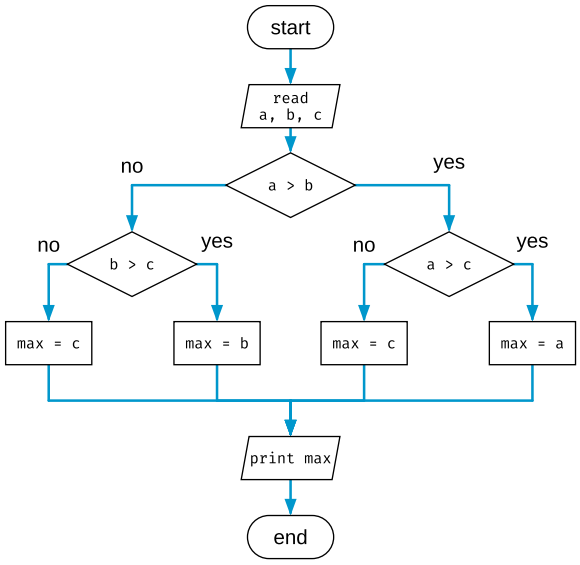

Fig. 28. Example of a flowchart created with Matofletcher.

CeTZ package has a built-in *tree* library that can be used, for example, to illustrate the merge sort algorithm (Fig. 29 shows the divide step).

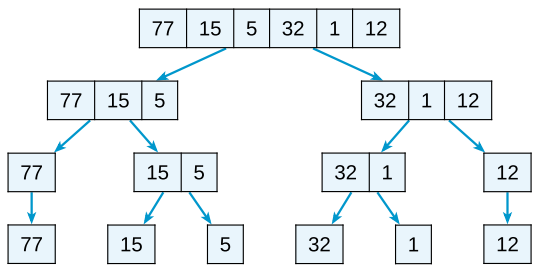

Fig. 29. Example of a tree diagram created with CeTZ's tree library.

The Diagraph package enables the inclusion of DOT diagrams [43] directly inside any document by using Wasm to render them without the need for an external software like Graphviz (Fig. 30).

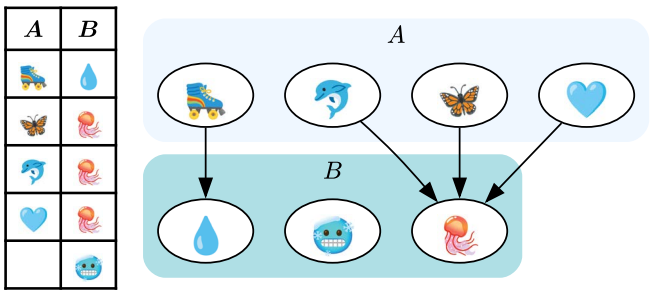

Fig. 30. Diagram of a Cartesian product of two emoji sets created with Wasm, Graphviz and the Diagraph package.

For creating truth tables, there is Truthfy package that can create a table from a logical expression. For hardware description and electronic processes related to it, there are several circuit diagram packages:

Zap draws electronic circuits that are aligned with IEC and IEEE/ANSI standards (using netlist-like semantics).

Circuiteria draws block circuit diagrams for a more abstract layer.

Quill draws quantum circuit diagrams with concise syntax.

B. Software

A prominent part of Computer Science is software and software engineering. For that, Typst has:

Built-in listings with automatic syntax highlighting (or custom parsers/themes), that can be enhanced with Codly (Fig. 32).

Built-in data loading functions for JSON, CSV, XML, etc. (or XLSX with ReXLlenT), for generating native tables or Lilaq graphs.

Sequence diagrams provided by Chronos (Fig. 31).

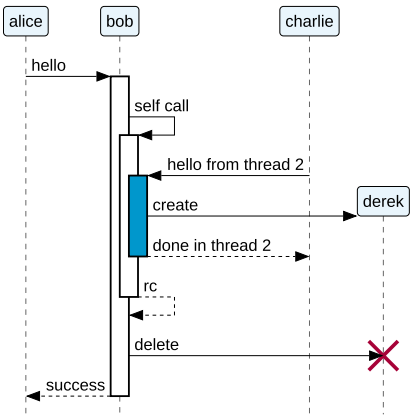

Fig. 31. Example of a sequence diagram created with Chronos.

```rust
fn area_of_circle(radius: f64) -> f64 {
    std::f64::consts::PI * radius * radius
}

fn main() {
    let area = area_of_circle(3.0);
    println!("A is {area}");
}
```

Rust

```javascript
function areaOfCircle(radius) {
  return Math.PI * radius * radius;
}

const area = areaOfCircle(3);
console.log(`Area is ${area}`);
```

JavaScript

Fig. 32. Styled sample codes in Rust and JavaScript (with Codly).

Activity, class, component, entity relationship and other diagrams can be created with Pintora text-to-diagram JavaScript library bundled in the Pintorita package. However, due to inherited Wasm limitations, these diagrams can significantly increase compilation time, i.e., up to several tens of seconds. As an example, Fig. 16 was created with this package.

Typst scripting language can tackle a lot of problems, such as prepro-cessing and visualizing data, implementing algorithms (sort, search, calendar-related, BNF-based recursive decent parser, Nassi–Shneider-man designs, etc.), generating raster images based on raw pixel data (supporting different pixel encodings), or even modifying SVG images with simple or regex-based substring replacements[31].

There are other several packages that ship with Wasm plugins for different language compilers and interpreters, such as:

Pyrunner for Python using RustPython,

Jogs (commented in Section III.B) for JavaScript using QuickJS, or

Matryoshka for running Typst inside Typst (without side effects).

Another notable limitation within Wasm-based packages is that network and input/output (I/O) operations do not work, but there is a possibility of passing project-local files to these packages. The second, and more impactful constraint, is their slower execution. It can be worth porting functions and algorithms to native Typst to significantly decrease compilation time, but to be sure it runs faster, profiling is necessary.

The Prequery package provides the ability to specify metadata regarding extra information associated with a document. This metadata then can be extracted with the `typst` query command to be used by, for instance, an external Python script. That way, the workflow requires one compilation without assets (to get the metadata), one run of the external preprocessor (i.e., the Python script) to gather the fetched metadata, and a second compilation with the necessary data already in place. This kind of preprocessing allows for fully automated creation of report-like documents (like the ones tackled in Section III.B).

Similarly, as stated in Section IV, a LaTeX-based math expression can be directly used as is through the MiTeX package. This can turn out helpful for migrating foreign .tex resources. Finally, with the Eqalc package, it is possible to convert a math expression to a native Typst function that can be evaluated and whose result can be plotted or tabulated (Fig. 33). This is a nice illustration of code reuse.

```
#import "@preview/eqalc:0.1.3": *
#let f = $g(t)=2t dot sqrt(e^t)+ln(t)+2pi$
#f
#math-to-table(f, min: 1, max: 3)
```

$$g(t) = 2t \cdot \sqrt{e^t} + \ln(t) + 2\pi$$

t	1	2	3
$g(t)$	9.58	17.85	34.27

Fig. 33. A math function $g(t)$ being translated to/evaluated by Typst.

C. Other Use Cases

The `eval` function allows writing the code only once while showing both the result and its associated source. Packages like Self-Example and IDWTET provide useful abstractions. Reproducibility often requires saving full source code or images. Typst's `pdf.attach` function allows embedding arbitrary byte sequences as files within a PDF, which can later be extracted if needed (research evidence, experimental assets, etc.).

From a management perspective, creating Gantt charts is possible with packages like Timeliney and Gantty (Fig. 34), while kanban board can be created with the Kantan package (Fig. 35).

[31] https://typst.app/universe/package/svgalpha

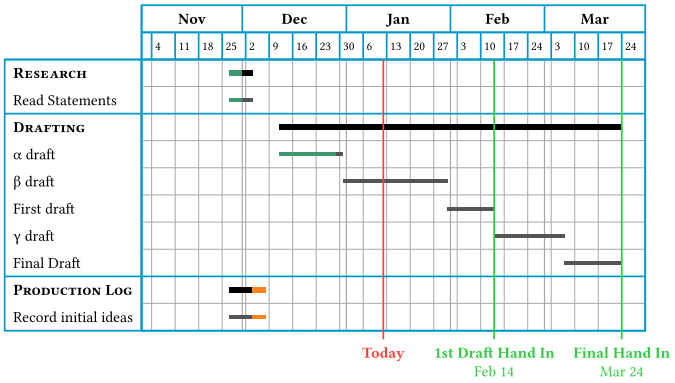

Fig. 34. Example of a Gantt chart designed with the Gantty package.

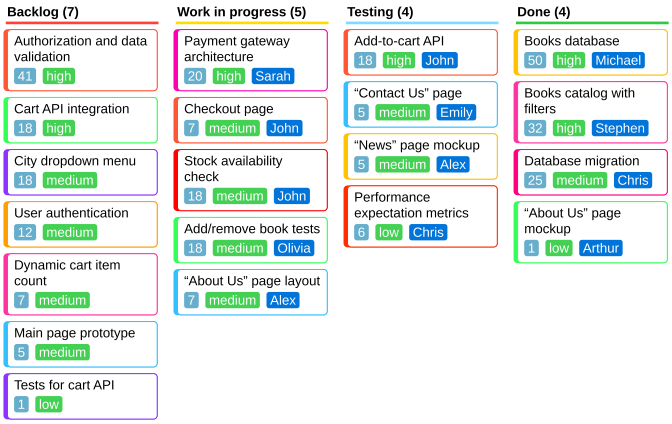

Fig. 35. Example of a kanban board made with the Kantan package.

As listed in Table IV, QR and bar codes can be issued and customized via the Tiaoma package, using a Wasm version of the Zint library.

XII. Slide Composition

Typst can be extended for slide creation through the Touying package, which provides a flexible framework similar to LaTeX's Beamer [44]. With Touying, users can design presentation slides directly in Typst, benefiting from its concise syntax, powerful layout capabilities, and smooth PDF output. The package supports themes, overlays, and struc-tured elements, making it easy to control visual style while focusing on content. Touying enables the creation of professional slides with minimal boilerplate, making it especially appealing for academic and technical presentations where precision and readability are key (Fig. 36).

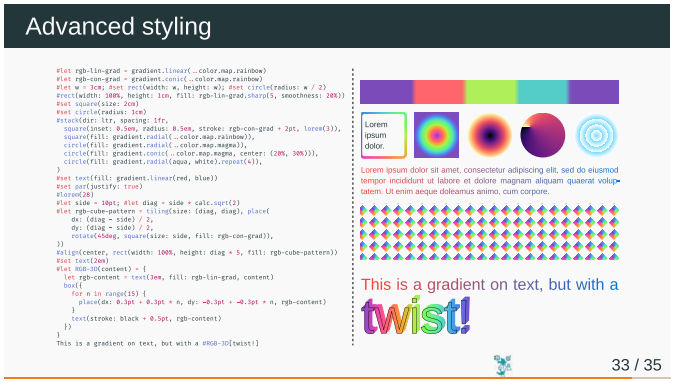

Fig. 36. A slide with complex content (code, gradients, advanced styling, etc.) created with the Touying package and the Metropolis theme.

XIII. Conclusions

Typst is a markup language for typesetting documents, combining ease of use and a rendering speed below the Doherty limit. It transforms plain text sources with markup into polished PDF files. Ideal for long-form writing, Typst excels at creating essays, articles, scientific papers, books, reports, and homework assignments. It also shines in technical fields, such as Mathematics, Physics, and Engineering thanks to its robust support for mathematical notation. Additionally, its powerful styling and automation capabilities make it perfect for document collections with a consistent design, like a book series or branded publications.

CRediT Authorship Contribution Statement

Andrey Voynov: Software, Validation, Writing – review & editing;

Alberto Corbi: Conceptualization, Project administration, Software, Supervision, Writing – original draft;

Pau López-Oliver: Software, Writing – review & editing;

David Gil: Formal analysis, Methodology, Software.

Data Statement

The research does not involve the use of external data. All figures and tables have been generated dynamically in Typst. The source code for this paper and instructions to reproduce the PDF can be found inside this PDF document in the form of attached files via the ISO 32000-1:2008 standard or at https://github.com/pammacdotnet/TypstPaper.

Declaration of Conflicts of Interest

We have no conflict of interest to declare.

Acknowledgment

Authors would like to express their gratitude to the Typst community, with a special focus on their two creators and main developers, Martin Haug and Laurenz MÃ¤dje, for their invaluable contributions. They would also love to thank the Editorial Board of IJIMAI and the Universidad Internacional de La Rioja (UNIR) for taking a leap of faith regarding the inclusion of Typst as a default mechanism to submit scientific papers.

References

[1] D. E. Knuth, *The Computers & Typesetting, Vol. A: The Texbook*. MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1986.

[2] L. Lamport, "LaTeX: a Document Preparation System." Addison-Wesley Reading, MA, USA, 1994.

[3] J. Voegler, J. Bornschein, and G. Weber, "Markdown – A Simple Syntax for Transcription of Accessible Study Materials," in *Computers Helping People with Special Needs: 14th International Conference, ICCHP 2014, Paris, France, July 9-11, 2014, Proceedings, Part I 14*, 2014, pp. 545–548.

[4] W. J. Doherty and W. G. Pope, "Computing as a tool for human augmentation," *IBM Systems Journal*, vol. 25, no. 3.4, pp. 306–320, 1986.

[5] B. Ewelina, P. Emilia, and W. Marcin, "LaTeX in Cloud—Unknown Tool in Technical Documents' Creation," in *Smart Innovations in Engineering and Technology*, 2020, pp. 75–89.

[6] B. Chagnon, "The Publishing Business: Desktop Publishing Software," *Journal of Electronic Publishing*, vol. 8, no. 1, 2002.

[7] D. Clark, "Content Management and the Separation of Presentation and Content," *Technical communication quarterly*, vol. 17, no. 1, pp. 35–60, 2007.

[8] T. Hassan and A. Hunter, "Knuth-Plass Revisited: Flexible Line-Breaking for Automatic Document Layout," in *Proceedings of the 2015 ACM*

Symposium on Document Engineering, 2015, pp. 17–20.
[9] D. E. Knuth and M. F. Plass, "Breaking paragraphs into lines," *Software: Practice and Experience*, vol. 11, no. 11, pp. 1119–1184, Nov. 1981.
[10] S. Feiner, "A Grid-Based Approach to Automating Display Layout," *Book Readings in Intelligent User Interfaces*, pp. 249–254, 1998.
[11] M. F. Plass, *Optimal Pagination Techniques for Automatic Typesetting Systems*. CA, USA: Stanford University, 1981.
[12] N. P. Rougier and B. Esfahbod, "Digital typography: 25 years of text rendering in computer graphics," *ACM SIGGRAPH 2018 Courses*. pp. 1–29, 2018.
[13] S. Toledo, "Typesetting Hebrew with LaTeX," *Eutupon*, vol. 6, pp. 39–56, 2001.
[14] A. Fisher, "Incremental algorithms for interactive text formatting," *Journal of Systems and Software*, vol. 16, no. 1, pp. 3–16, Sept. 1991.
[15] M. Elkhayati, Y. Elkettani, and M. Mourchid, "Segmentation of Handwritten Arabic Graphemes Using a Directed Convolutional Neural Network and Mathematical Morphology Operations," *Pattern Recognition*, vol. 122, 2022.
[16] D. Barron and M. Rees, *Text processing and typesetting with UNIX*. MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1987.
[17] S. J. DeRose, *The SGML FAQ book: Understanding the Foundation of HTML and XML*, vol. 7. NY, USA: Springer Science+Business Media, 1997.
[18] T. W. Cole, W. H. Mischo, R. Ferrer, and T. G. Habing, "Using XML, XSLT, and CSS in a Digital Library," in *Proceedings of The Annual Meeting-American Society for Information Science*, 2000, pp. 430–439.
[19] M. Dominici, "An overview of Pandoc," *TUGboat*, vol. 35, no. 1, pp. 44–50, 2014.
[20] M. Pégourié-Gonnard, "A guide to LuaLaTeX," *DANTE e.V.*, vol. 5, 2013.
[21] F. Mittelbach, "LaTeX's hook management." 2024.
[22] Y. Xie, "knitr: A Comprehensive Tool for Reproducible Research in R," *Implementing reproducible research*. Chapman, Hall/CRC, pp. 3–31, 2018.
[23] F. Leisch, "Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis," in *COMPSTAT: Proceedings in computational statistics*, 2002, pp. 575–580. doi: https://doi.org/10.1007/978-3-642-57489-4_89.
[24] B. Baumer and D. Udwin, "R Markdown," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 7, no. 3, pp. 167–177, 2015.
[25] J. Bettels and F. A. Bishop, "Unicode: A Universal Character Code," *Digital Technical Journal*, vol. 5, no. 3, pp. 21–31, 1993.
[26] S. Kottwitz, "LaTeX Graphics with TikZ: A practitioner's guide to drawing 2D and 3D images, diagrams, charts, and plots," 2023, *Packt Publishing, Birmingham, UK*.
[27] S. Tosi, *Matplotlib for Python Developers*, vol. 307. Birmingham, UK: Packt Publishing, 2009.
[28] A. Corbi, D. Burgos, and A. M. Pérez, "Cloud-Operated Open Literate Educational Resources: The Case of the MyBinder," *IEEE Transactions on Learning Technologies*, vol. 17, pp. 893–902, 2023.
[29] D. Datta, "Bibliography with the BIBTeX Program," *LaTeX in 24 Hours: A Practical Guide for Scientific Writing*, pp. 141–151, 2017.
[30] M. Fenner, K. Scheliga, and S. Bartling, "Reference management," *Opening science: The Evolving Guide on How the Internet is Changing Research, Collab-oration and Scholarly Publishing*, pp. 125–137, 2014.
[31] M. E. Haug, "Fast Typesetting with Incremental Compilation," 2022. doi: https://doi.org/10.13140/RG.2.2.15606.88642.
[32] L. Mädje, "A Programmable Markup Language for Typesetting," 2022. [Online]. Available: https://laurmaedje.github.io/programmable-markup-language-for-typesetting.pdf
[33] S. Klabnik and C. Nichols, *The Rust Programming Language, 2nd Edition*. CA, USA: No Starch Press, 2023.
[34] A. Haas *et al.*, "Bringing the web up to speed with WebAssembly," in *Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation*, 2017, pp. 185–200. doi: https://doi.org/10.1145/ 3062341.3062363.
[35] P. K. Janert, *Gnuplot in Action: Understanding Data with Graphs*. NY, USA: Simon and Schuster, 2016.
[36] G. Lacombe, K. Masalygina, A. Tahiri, C. Adam, and C. Lauradoux, "Can You Accept LaTeX Files from Strangers? Ten Years Later," *arXiv preprint arXiv:2102.00856*, 2021.
[37] S. Kim and J. Lee, "Analysis of Security Vulnerabilities and Personal Resource Exposure Risks in Overleaf," *Journal of The Korea Society of Computer and Information*, vol. 29, no. 7, pp. 109–115, 2024.
[38] A. Lombardi, *WebSocket: Lightweight Client-Server Communications*. CA, USA: O'Reilly Media, Inc., 2015.
[39] A. Chaleplioglou and A. Koulouris, "Preprint paper platforms in the academic scholarly communication environment," *Journal of Librarianship and Infor-mation Science*, vol. 55, no. 1, pp. 43–56, 2023.
[40] J. Wright, "siunitx: A comprehensive (SI) units package," *TUGboat*, vol. 32, no. 1, pp. 95–98, 2011.
[41] M. E. Peskin and D. V. Schroeder, *An Introduction to Quantum Field Theory*. FL, USA: CRC Press, 2018. doi: https://doi.org/10.1201/9780429503559.
[42] G. Giachetta, *Advanced Classical Field Theory*. Singapore: World Scientific, 2009.
[43] E. R. Gansner, "Drawing graphs with Graphviz," *Technical report, AT&T Bell Laboratories, Murray, Tech. Rep, Tech. Rep.*, 2009.
[44] M. Hofert and M. Kohm, "Scientific Presentations with LaTeX," *The PracTEX Journal*, no. 2, 2010.

Andrey Voynov

Andrey Voynov is a master's student in software engineering with a strong passion for programming and free/open-source software. He began coding in QBasic and Pascal during ele-mentary school, actively competed in Informatics Olympiads, and earned a 2nd-degree diploma at the 2020 All-Russian Forum of Scientific Youth. Andrey has created hundreds of projects, contributed to many others, and since discovering Typst in 2023, uses it exclusively for documents and slides, promoting its adoption in Russia through teaching, curriculum updates, and feature contributions.

Alberto Corbi

Alberto Corbi is a tenured professor at the Universidad Internacional de La Rioja (UNIR). He is currently involved in a variety of research fields: e-learning standards, systems interoperability, medical physics, radiological protection, STEAM education, the social implications of technology (with emphasis on social networks), e-health advancement. He has published over 20 research papers on all these subjects, and he is a speaker and knowledge disseminator on radios, podcasts, scientific workshops, magazines, academic settings, and outreach events.

Pau López-Oliver

Pau López-Oliver is a theoretical physicist with a M.Sc. in Advanced Physics from Universitat de València and a B.Sc. in Physics from the University of Murcia. His research centers on understanding particle creation in quantum field theory in curved and dynamical backgrounds. He is dedicated to advancing fundamental physics while promoting science communication. He currently teaches in the Physics degree program at the Universidad Internacional de La Rioja (UNIR).

David Gil

David Gil works at the Spanish weather service (AEMET) as an IT Specialist. He is currently pursuing a bachelor's degree in Physics at the Universidad Internacional de La Rioja (UNIR) with the goal of becoming a meteorologist and work-ing with Numerical Weather Prediction systems. His work involves programming (preferring functional programming languages), high-performance computing, and developing web applications that deliver weather forecast products to forecasters. He is also engaged in projects related to containerization technologies and workflow automation, aiming to bridge computer science and meteorology.