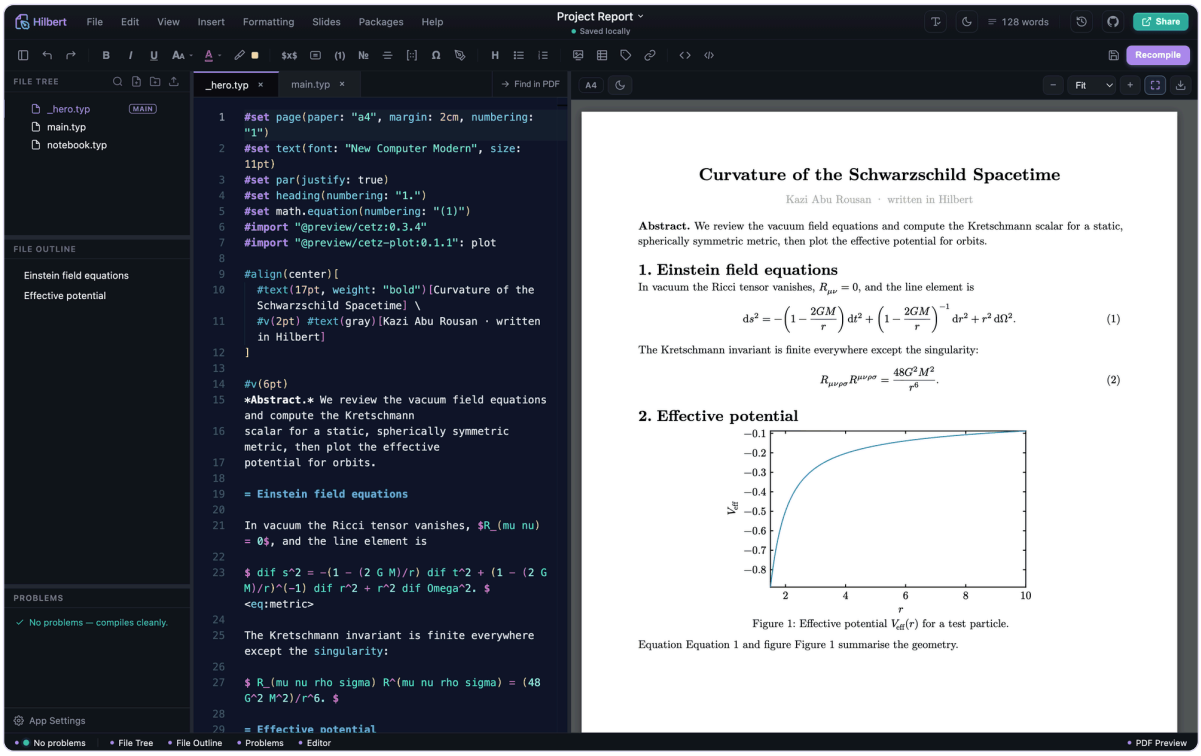
Hilbert

An offline scientific-writing IDE for Typst

Handbook for version 0.2.3 · 26 August 2026



This handbook covers two things: how to get Hilbert working properly on your machine, one step at a time, and everything the application can do once it is. Chapters 1 to 4 are the setup; chapters 5 to 13 are the features, grouped by what you are trying to write; chapter 14 onwards is reference material you can skip until you need it.

Contents

CHAPTER 1

What Hilbert is, and what it needs

Hilbert is a desktop editor for Typst: a code editor on the left, a live PDF on the right, and the syntax you would otherwise memorise sitting in menus. It runs entirely on your own machine, works with no network at all, and can execute Python, Julia or Wolfram code and put the result straight into the document.

It does not reimplement Typst. It drives the real compiler, which means two external tools matter before anything else works.

1.1 The two tools Hilbert drives

Tool	What it does for you	Required?
Typst CLI 0.15+	Compiles the document and produces the PDF you see in the preview. Nothing renders without it.	Yes
tinymist	The Typst language server: hover documentation, autocomplete, live errors and warnings, go-to-definition, rename, formatting.	Recommended

Without tinymist the editor still compiles and previews perfectly well; the language-server features simply stay quiet. Without the Typst CLI, Hilbert opens and then cannot render anything, which is far and away the most common "it does not work" report.

1.2 Optional, only if you want to run code

- **Python 3** with `numpy`, `matplotlib` and `sympy` — code cells, plots, and the simplify/solve/integrate tools.
- **Julia**, plus `Latexify` if you want equation mode to typeset results.
- **WolframScript**, if you have Mathematica.

None of these are needed to write a document. Set them up when you first want to compute something, and skip to Section 4 when you do.

1.3 Hardware and platform

Hilbert is small: the backend idles at about 12 MB and installs in roughly 37 MB, and a thousand-file project takes it to about 18 MB. Anything that runs a modern browser runs this. It is packaged for Windows 10 and 11, macOS on both Apple Silicon and Intel, and Linux as AppImage, `.deb` and `.rpm`.

CHAPTER 2

Installing Hilbert

Pick your platform below and follow it in order. Every route ends at the same application; the package managers simply keep it updated for you.

2.1 Windows

Hilbert is in the Microsoft package repository, which is the least fiddly route and installs Typst and tinymist at the same time.

1. Open Terminal, PowerShell or Command Prompt.
2. Install the application:

```
winget install --exact --id Aburousan.Hilbert
```

3. Install the compiler and the language server:

```
winget install --exact --id Typst.Typst
winget install --exact --id Myriad-Dreamin.Tinymist
```

4. Close the terminal and open Hilbert from the Start menu.

> `--exact` matters for tinymist: three packages published by the same author begin with the same identifier, and without it winget cannot tell which one you meant.

Everything installs per user, so no administrator rights are involved. If you would rather not use a package manager, download the `.exe` (or `.msi`) from the Releases page and run it — it is the same application, and it will still find a Typst or tinymist installed by winget, scoop or chocolatey afterwards. Later, `winget upgrade --id Aburousan.Hilbert` picks up a new release, though the app also updates itself.

2.2 macOS

1. Download the right disk image from the Releases page: `…-macOS-arm64.dmg` for M-series Macs, `…-macOS-x64.dmg` for Intel ones. *About This Mac* tells you which you have.
2. Open the `.dmg` and drag Hilbert into Applications.
3. Install the compiler and the language server:

```
brew install typst
brew install tinymist
```

4. Launch Hilbert from Applications.

> **First launch will probably be refused.** The app is not notarised — that needs a paid Apple developer account — so macOS quarantines it, and moving or renaming the `.app` can break its ad-hoc signature. If it will not open, or claims to be *damaged*, run these *as two separate commands*, one per line:
>
> ```
> xattr -cr "/Applications/Hilbert.app"
> codesign --force --deep --sign - "/Applications/Hilbert.app"
> ```

> Pasted onto a single line, the shell reads `--force` as an option to `xattr` and reports it as unrecognised. This is a one-time step.

There is also a Homebrew cask: `brew install --cask aburousan/hilbert/hilbert`, then `brew upgrade --cask hilbert` later. Gatekeeper may still refuse the first launch; either add `--no-quarantine` to the install or run the two commands above.

2.3 Linux

The apt repository is the route that keeps itself current on Debian and Ubuntu.

1. Add the signing key and the repository:

```
curl -fsSL https://aburousan.github.io/hilbert-apt/hilbert-archive-keyring.asc \
  | sudo gpg --dearmor -o /usr/share/keyrings/hilbert-archive-keyring.gpg
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/hilbert-archive-keyring.gpg]
https://aburousan.github.io/hilbert-apt stable main" \
  | sudo tee /etc/apt/sources.list.d/hilbert.list
```

2. Install it:

```
sudo apt update && sudo apt install hilbert
```

3. Install Typst and tinymist. Your distribution may package them; otherwise `cargo install typst-cli` and `cargo install tinymist`, or take the release binaries and put them on your `PATH`.

From then on `apt upgrade` keeps Hilbert current along with everything else.

On other distributions, take the `.AppImage` (which auto-updates itself) or the `.rpm`. Make an AppImage executable with `chmod +x` before running it.

> If you intend to run code cells on Linux, install `bubblewrap` too — see Section 4. Without it, Hilbert will still run code, but unconfined.

2.4 Confirming the install is sound

Open Hilbert and check three things before you write anything real:

1. **Help → Features & Help** opens: the window is alive.
2. **App Settings → General** names a Typst path and version, and says whether tinymist is installed and running. This is the screen that answers "why does nothing compile".
3. Type a line into a document and watch the preview appear on the right.

If the second step comes up empty, the tools are not on the `PATH` that Hilbert inherited — see Section 24.4, which is a more common problem than it sounds.

2.5 Keeping it updated

The desktop build checks for a new version on launch and asks before installing anything; if the check cannot run, the app starts normally regardless. On Linux the AppImage auto-updates and the `.deb` follows the apt repository. If you installed through winget or Homebrew, their `upgrade` commands work too.

CHAPTER 3

Your first project

Hilbert is folder-based in the way VS Code is: you open a directory and everything in it becomes the project.

3.1 Opening a workspace

1. **File → Open Folder as Workspace**, and pick a folder — an existing paper, or a new empty directory.
2. The file tree appears on the left. **File → New File** makes `main.typ` if you need one.
3. Type. The PDF on the right recompiles after a short pause; `⌘S` saves and recompiles immediately.

Your documents live in `~/Documents/Hilbert` by default, and **File → Open Recent** brings back anything you have had open before.

3.2 Multi-file documents

Split a long piece across files and `#include` them from the root document. Hilbert always compiles from the project root — `main.typ`, or whatever `typst.toml` names as the entrypoint — so included chapters that share a bibliography or cross-reference each other's labels render as one whole rather than as fragments.

The root file carries a MAIN badge in the tree. To move it, right-click any `.typ` file and choose **Set as main file**.

3.3 The panels

The View menu and the status bar along the bottom switch each panel on and off independently: file tree, outline, problems, editor, preview. Hide the editor to read; hide the preview to write. Panes resize by dragging, and **File → New Window** opens a second project in its own window — one application with one Dock or taskbar icon, but independent previews.

3.4 Themes

Five interface themes dress the whole window, not merely the editor pane: **Ink** (the default charcoal), **Paper**, **Sepia** for long low-blue sessions, **Midnight** for a dark room or an OLED panel, and **High Contrast**. Cycle them with the sun/moon button in the header, or pick one in **App Settings → Interface theme**. The PDF preview keeps its own light/dark toggle, because the page itself does not always want to match.

3.5 App Settings

App Settings sits at the bottom of the sidebar, and in `⌘K`. It has four panels: General, Interpreters, Git & GitHub, and Cloud Accounts.

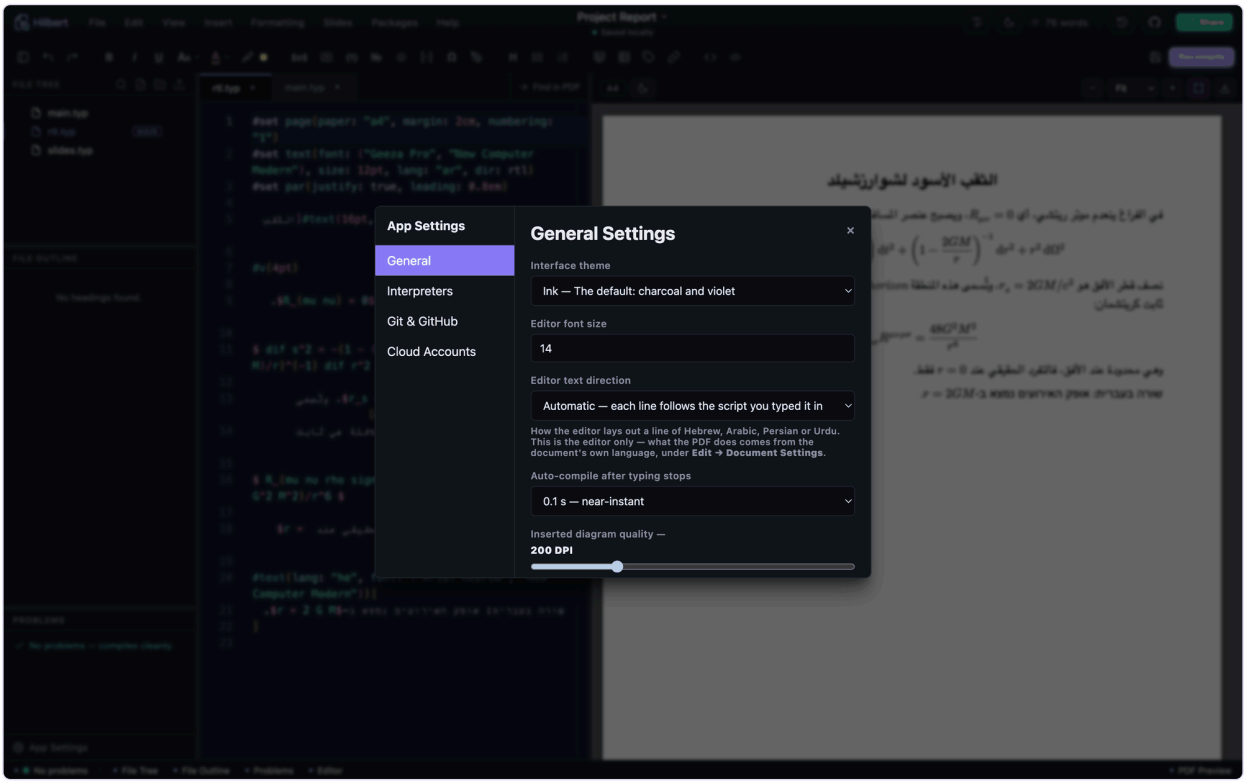


Figure 1: General settings: theme, editor font size, how the editor lays out right-to-left lines, how long it waits before recompiling, and the resolution inserted diagrams are rendered at.

Two settings there are worth changing early. **Auto-compile after typing stops** runs from near-instant to several seconds — shorten it on a small document, lengthen it if you are working on something big enough that compiling competes with your typing. **Inserted diagram quality** is the DPI that the visual builders rasterise at, which trades file size against how sharp a diagram looks when a reader zooms in.

3.6 What is remembered

The interface theme, editor font size, auto-compile delay, which panels are showing, the pane sizes and the interpreter chosen for each language are stored next to the session on disk, so they survive a restart, a reboot and a second window.

They used to live in the webview's storage, which is tied to whichever port the application happened to get — and losing that port meant the editor reopening at 14 pt as though you had never told it otherwise.

CHAPTER 4

Setting up code execution

Skip this chapter until you want Hilbert to compute something. Nothing here is needed to write and compile a document.

4.1 Choosing an interpreter

1. Open **App Settings → Interpreters**.
2. Hilbert lists the Python, Julia and Wolfram installations it found, conda environments included.
3. Pick the default for each language. The choice is remembered across restarts.

If the environment you want is missing, it was not on the `PATH` the application inherited; Section 24.4 explains why that happens and how to fix it.

4.2 Choosing how figures come out

The same settings panel controls the format plots are saved in. PNG is the default; switch it to SVG or PDF and plots stay vector, which prints sharp at any size.

Asking for a format inside the code always wins over the setting — Julia's `plot(x; fmt = :pdf)`, or naming the file yourself with `savefig("figure.svg")`. EPS is supported for journals that insist on it: Typst cannot embed EPS, so those runs also write a PDF of each figure and the document points at that one.

4.3 The sandbox

Where the kernel can confine a run, Hilbert makes it: bubblewrap on Linux, Seatbelt on macOS. Confined code can write to its own scratch directory and nowhere else, has no network, and cannot read `~/.ssh`, `~/.gnupg`, `~/.aws` or the other credential directories. Figures still reach your document, because the application copies them out afterwards. On Linux the run also gets its own process, IPC and hostname namespaces.

On Linux, install bubblewrap to get this:

```
sudo apt install bubblewrap      # Debian / Ubuntu
sudo dnf install bubblewrap      # Fedora
```

Where no sandbox exists — Windows, or `HILBERT_SANDBOX=off` — a pattern screen refuses process, network, shell and destructive calls instead. **App Settings → Interpreters** states which of the two is in force. Wolfram is never confined: `wolframscript` launches a separate kernel over a loopback socket and picks that kernel from state outside the run directory, so confining it either stops it starting or silently switches Mathematica versions. It keeps the pattern screen.

> A sandbox is a real boundary, but for documents you genuinely do not trust it is not the only one worth having. A container or a VM assumes less and costs you nothing.

4.4 Turning it off

`ALLOW_CODE_EXECUTION=0` in the environment disables code execution completely. `EXEC_TIMEOUT_MS` changes the per-run wall-clock limit from its 45-second default. The full list is in Section 21.

4.5 Checking it works

Put this in a document and press the `</>` button in the toolbar:

````
```python
from math import sqrt
print(f"the golden ratio is {(1 + sqrt(5)) / 2:.6f}")
```
````

The output appears below the block, and the compiled PDF badges the block with the Python logo. If nothing happens, the interpreter produced no output — check **App Settings → Interpreters** first.

CHAPTER 5

Writing: the editor and the preview

5.1 The editor

The editor is Monaco, the same component VS Code is built on, with Typst syntax highlighting on top. With tinymist running you also get:

- hover documentation and signatures for any function;
- autocomplete for every builtin, package export and label;
- `@`-reference completion, and image-path completion inside `image("…")`;
- control-flow completions offering both the `{ }` code body and the `[ ]` content body for `if`, `for` and `while`;
- live errors, warnings, information and hints, both inline and in the Problems panel.

The Edit menu and the editor's right-click menu expose the rest of the language server: go to definition, find references, rename a symbol across the file (`F2`), quick fixes, and whole-document formatting with the bundled typstyle formatter.

Comment or uncomment the current line or selection with `⌘/` — it understands Typst, Python, Julia, `.bib` and more.

5.2 The preview

The PDF recompiles as you type. It has zoom, fit-to-width, a dark PDF mode, and double-click-to-source, which reads the words around where you clicked so it lands on the right occurrence rather than the first one.

When a compile fails you keep the last good preview. The errors move to their own Problems tab and a slim strip appears at the bottom of the preview; click it for the full list. A typo mid-sentence never blanks the page, and the last good render is up from the moment you open the project.

There is also an experimental **HTML Preview** in the View menu, which renders the document through Typst's HTML export.

5.3 The Problems panel

Errors, warnings, information and hints all land in one clickable list — click an entry to jump to the line. Typst's own compile errors arrive there alongside tinymist's analysis, and the count in the status bar tells you at a glance whether the document compiles.

While a compile is running the previous PDF stays on screen. Past a few seconds the status bar says **still waiting on Typst**, and saving keeps working meanwhile.

5.4 The word count

The status bar counts the words of the *rendered* document — it reads them from the PDF, so `#set` rules and `#import` lines do not inflate the number the way a source-text count would.

5.5 The command palette

`⌘K` opens a palette covering every menu action in the application, searchable by name. It is the fastest route to anything in this handbook, and the full catalogue is listed in Section 20.

CHAPTER 6

Files and projects

6.1 The file tree

Multi-select, drag-and-drop moves, rename, duplicate, delete, cut, copy, paste, a right-click menu, new file and folder, asset upload, compress to `.zip`, and reveal-in-file-manager. Full-text search across the whole workspace jumps you straight to the matching line.

6.2 When files change underneath you

Files changed on disk by Git or another editor are picked up automatically. If you had unsaved edits in the same file, Hilbert shows both versions side by side and asks which you want, rather than silently discarding either.

6.3 Version history

View → Version History keeps earlier states of the document so you can look back without a commit.

6.4 Importing into a project

File → Import Folder into Project copies a directory in. **File → Import Font** takes a `.ttf` or `.otf` and wires up the `#set text(font: "…")` line for it.

CHAPTER 7

How fast it is, and why

Every number here comes from `docs/PERFORMANCE.md` in the repository, which is reproducible: `node scripts/bench.mjs` generates the workspaces and runs the backend against them, and `python scripts/bench_plot.py` draws the chart.

7.1 The backend, across project sizes

Four generated workspaces, where `main.typ` `#include`s every chapter, so a full compile scales with the whole project.

Metric	Tiny 5 files	Medium 32	Thesis 202	Huge 1002
Index the file tree (ms)	0.9	0.6	1.6	4.4
Full-text search, average (ms)	0.4	1.1	3.9	17.3
Full-text search, worst (ms)	0.6	1.9	5.7	18.4
Create, rename, delete a file (ms)	1.7	0.8	0.8	0.7
Full compile (ms)	212	134	204	536
Memory at start (MB)	12	12	12	12
Memory after 100× load (MB)	14	14	15	18

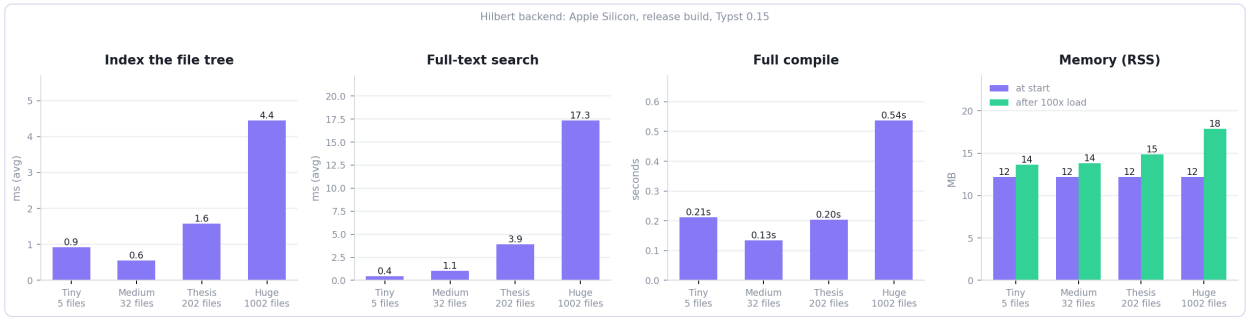


Figure 2: The same figures as a chart. Search is the only thing that grows visibly with project size.

The backend starts at **12 MB** and stays there: a thousand-file project costs about 6 MB more. Search is the only measurement that scales visibly, and even across 1000 files its worst case is 18 ms — inside a single keystroke. File operations stay under 2 ms at every size. Compile time is dominated by the Typst CLI rather than by Hilbert, which is why the thousand-chapter figure of 0.54 s sits close to what `typst compile` costs on its own.

7.2 Starting up

Metric	Value
Backend process to first HTTP response	32 ms warm, about 650 ms cold
App launch to embedded server ready	231 ms
Page load to an interactive editor	about 300 ms
Page load to a rendered 300-page PDF	about 1.5 s
Installed size of `Hilbert.app`	37 MB — 16 MB binary, 18 MB UI, 2 MB packages
Frontend JS heap	about 31 MB

A 300-page stress document — 7000 lines, 300 tables, 100 code blocks, heavy maths — compiles in about **1.0 s** and first-renders in about 1.5 s. Twenty consecutive compiles of it leave resident memory flat.

The live preview keeps one `typst watch` process per workspace and entry file, so warm edits reuse the compiler's state instead of paying for a new process each time.

7.3 A real paper, on two very different machines

The generated workspaces vary one thing at a time. This is a single physics note somebody actually wrote: 1365 lines, 62 KB, 11,395 words, 42 numbered equations, 12 figures, 47 sections.

Metric	Apple M2 laptop	Xeon E3-1220 v5 server
Typst compile, warm	**0.26 s**	**0.66 s**
Typst compile, cold	0.67 s	0.93 s
Proofread the whole paper, first pass	0.74 s	1.4 s
Proofread after one word is typed	**11 ms**	—
Issues found	300	300
Backend memory after proofreading	257 MB	253 MB

A four-year-old Xeon lands within a factor of two of the M2 on every figure and uses the same memory. The work is single-threaded and cache-friendly, so a bigger machine buys very little — 8 GB of RAM is enough, and the 250 MB is dictionaries.

7.4 Where the memory goes

Process	Resident memory
Backend, idle	12 MB
Backend once proofreading is switched on	about 174 MB, 257 MB after checking a 62 KB paper
WebKit content process	about 200 MB
tinymist language server	about 33 MB

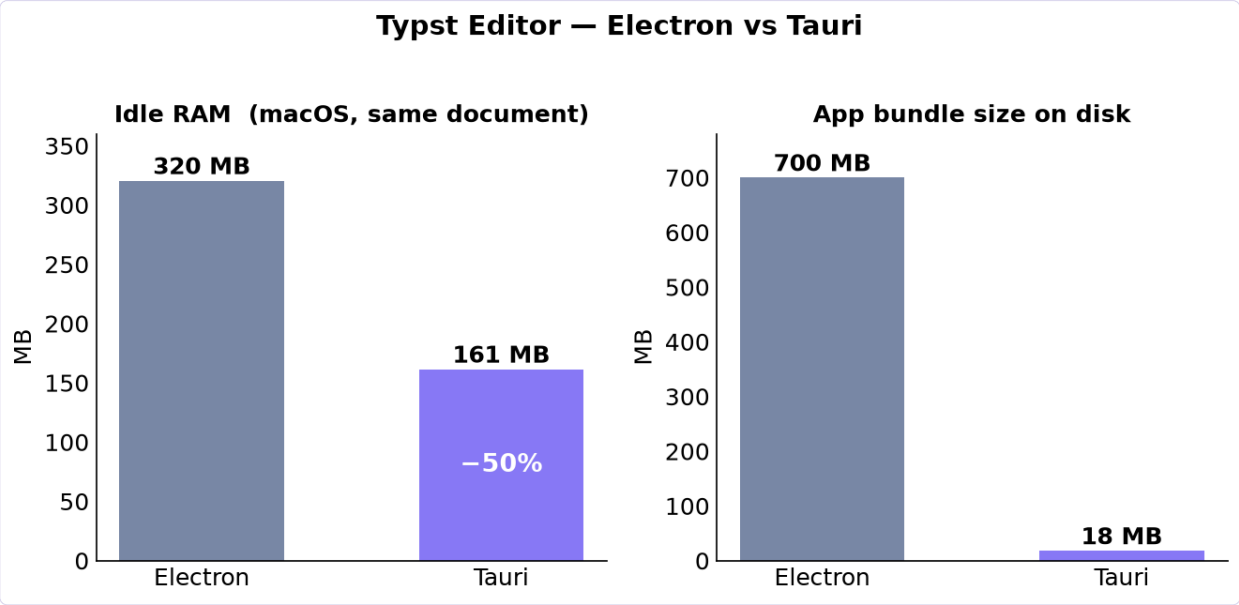


Figure 3: Against the Electron edition this application replaced.

The WebView dominates, which is the expected shape for a Tauri application: the system WebKit is shared rather than bundled, so the binary stays at 16 MB where an Electron build ships a copy of Chromium. The earlier Electron edition of this same application idled around 320 MB across five processes and unpacked to 711 MB on disk.

Switching proofreading on is the largest single memory decision in the application, and it belongs to you: the dictionaries load the first time proofreading actually runs, not at launch, which is what took idle memory from 173 MB down to 12 MB.

7.5 Two optimisations worth knowing about

Proofreading looks at what changed. A full pass over the paper above costs about 740 ms. The document is cut into pieces at blank lines, each piece's answers are kept under a hash of that piece, and only changed pieces are rechecked — so editing one paragraph costs one paragraph: **11 ms instead of 740**, a factor of sixty-five. Cuts are only allowed where a piece parses the same alone as it did in the document, which means never inside a raw block or a display formula.

Slide Studio does not redraw the deck when you move a box. Dragging one element used to rewrite the whole thumbnail rail, so the cost of moving something grew with the length of your deck. Each rail row is now memoised on its slide, and a drag costs the same whatever the deck's length:

Slides	Before	After	Worst frame, before → after
6	6.0 ms	5.7 ms	6.9 → 5.1 ms
24	6.8 ms	6.2 ms	11.2 → 5.9 ms
60	9.6 ms	6.3 ms	24.6 → 7.1 ms
120	14.9 ms	6.5 ms	40.2 → 7.9 ms

At 120 slides a frame used to take 14.9 ms of the 16.7 available at 60 Hz, and the worst took 40 — three frames on the floor.

CHAPTER 8

Inserting the annoying parts

Everything in this chapter is in the Insert menu and in ⌘K.

8.1 Document furniture

Title blocks, authors, institutes, abstracts, headings, and theorem/proof/lemma blocks — plain or in coloured boxes, each kind numbered separately.

8.2 Equations

Inline (⌘E), block, multiline/aligned, and numbered (⌘⇧E) equations, with numbering on by default. Toggle numbering for the equation under the cursor with ⌘⇧N, or for the whole document from the Edit menu. Equation templates (⌘⇧G) are fill-in skeletons for the shapes you write most often.

Also here: conditionals and piecewise `cases`, over- and under-braces, and cancel/strike terms.

8.3 Matrices

Matrix Studio (⌘⇧M) is a visual grid — type into the cells, set the delimiter, add augmentation lines with a chosen colour, and insert clean Typst.

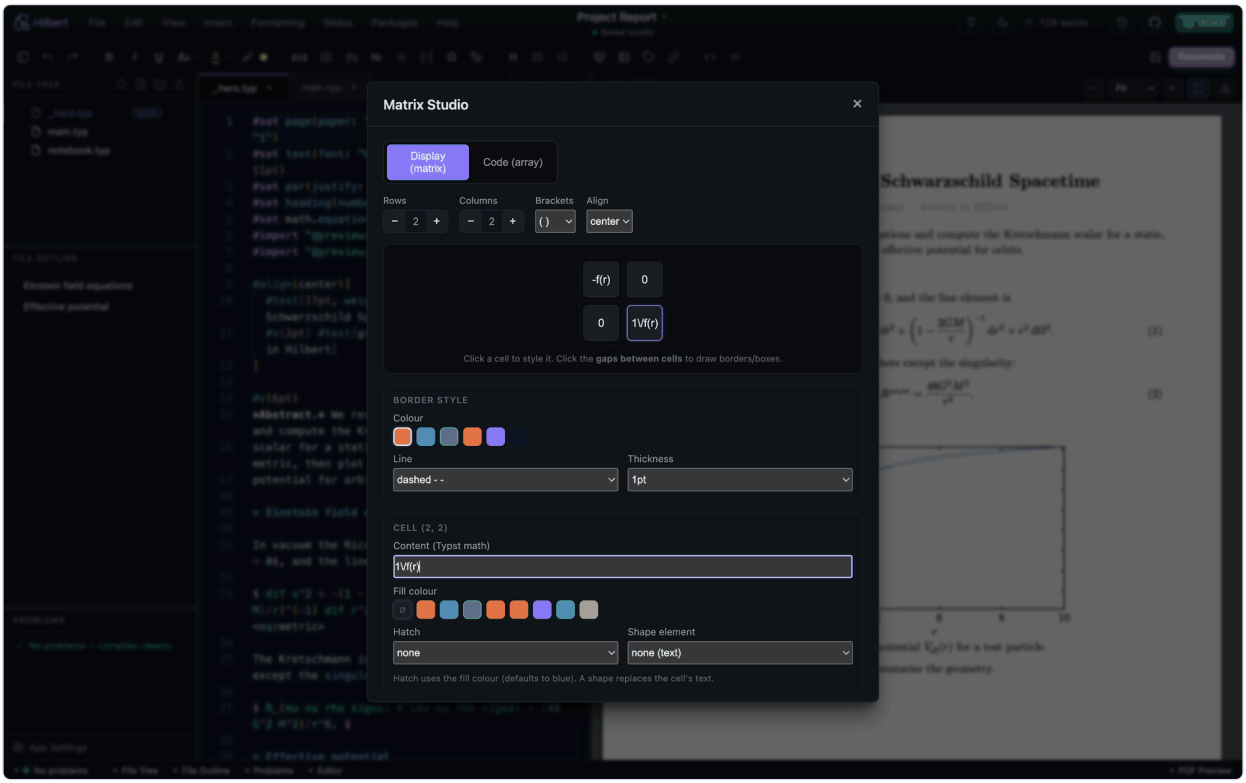


Figure 4: Matrix Studio, with an augmentation line configured.

8.4 Lists, notes and text blocks

Bullet, numbered, nested and term/definition lists. Callout and admonition boxes, block quotes, footnotes, margin notes, and a full-width horizontal rule (⌘⇧H).

8.5 Tables, figures and images

Tables with per-column widths and alignment, figures, images, and subfigures placed side by side with `a` / `b` labels. Most of these carry a *centre on page* toggle. **Place Image** handles wrapping text around a figure, floating it, or dropping it below. The built-in image editor crops and rotates PNGs and JPEGs before inserting; SVGs open as a safe preview.

A whiteboard/sketch tool (Excalidraw) is in the same menu for hand-drawn figures.

8.6 Page setup and text formatting

Formatting → Page Setup writes the `#set page(...)` rule for paper size, per-side margins, header, footer and page numbers.

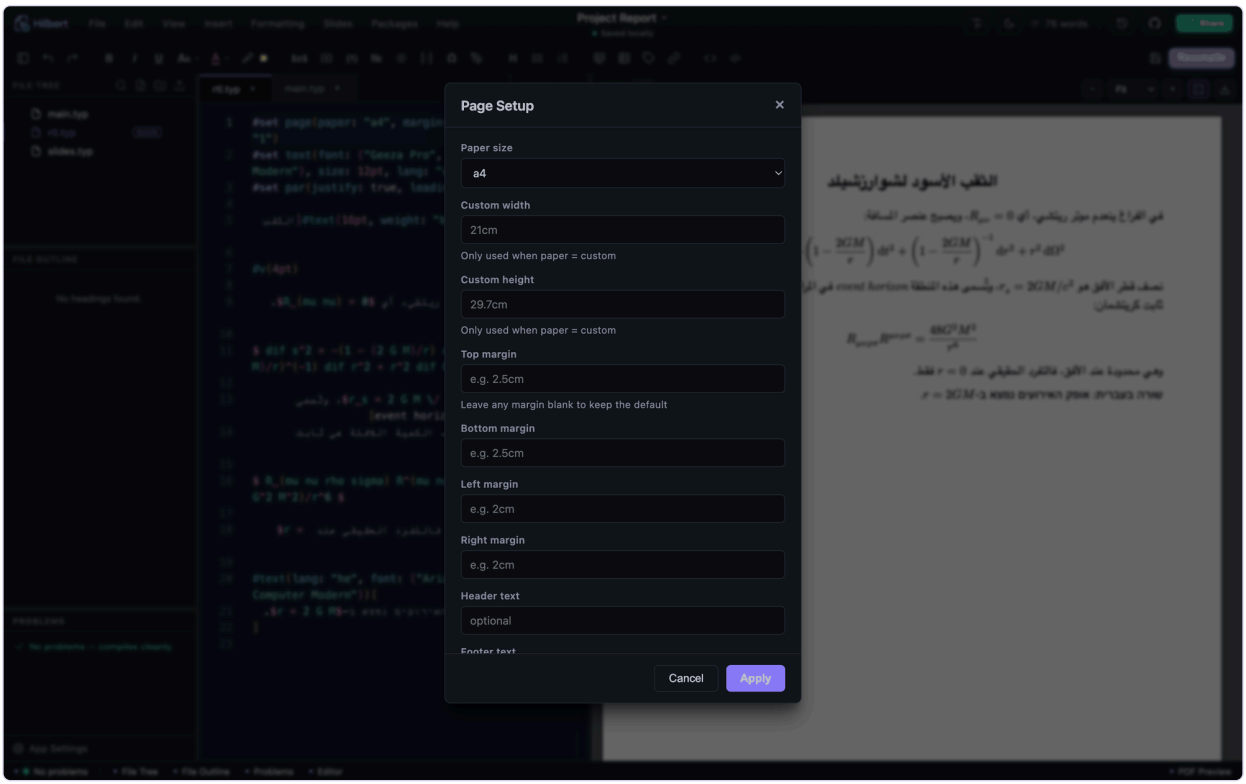


Figure 5: Page Setup. Any margin left blank keeps Typst's default rather than being forced to a number.

Text formatting covers bold (`⌘B`), italic (`⌘I`), underline, super- and subscript, a draggable colour picker, highlight, strike-through, boxed selections with fill, border and texture, a font-size dropdown, alignment, rotation, small caps, letter spacing, and non-breaking spaces.

Right-to-left text is handled explicitly: keep a selection RTL or LTR, keep it together as a direction isolate, or flip the direction of the current line.

CHAPTER 9

Writing right to left

Arabic, Hebrew, Persian, Urdu and the rest are not an afterthought here. Typst itself sets the document's direction; what Hilbert adds is an editor that lays out mixed source correctly, and the controls for the lines where the automatic rule guesses wrong.

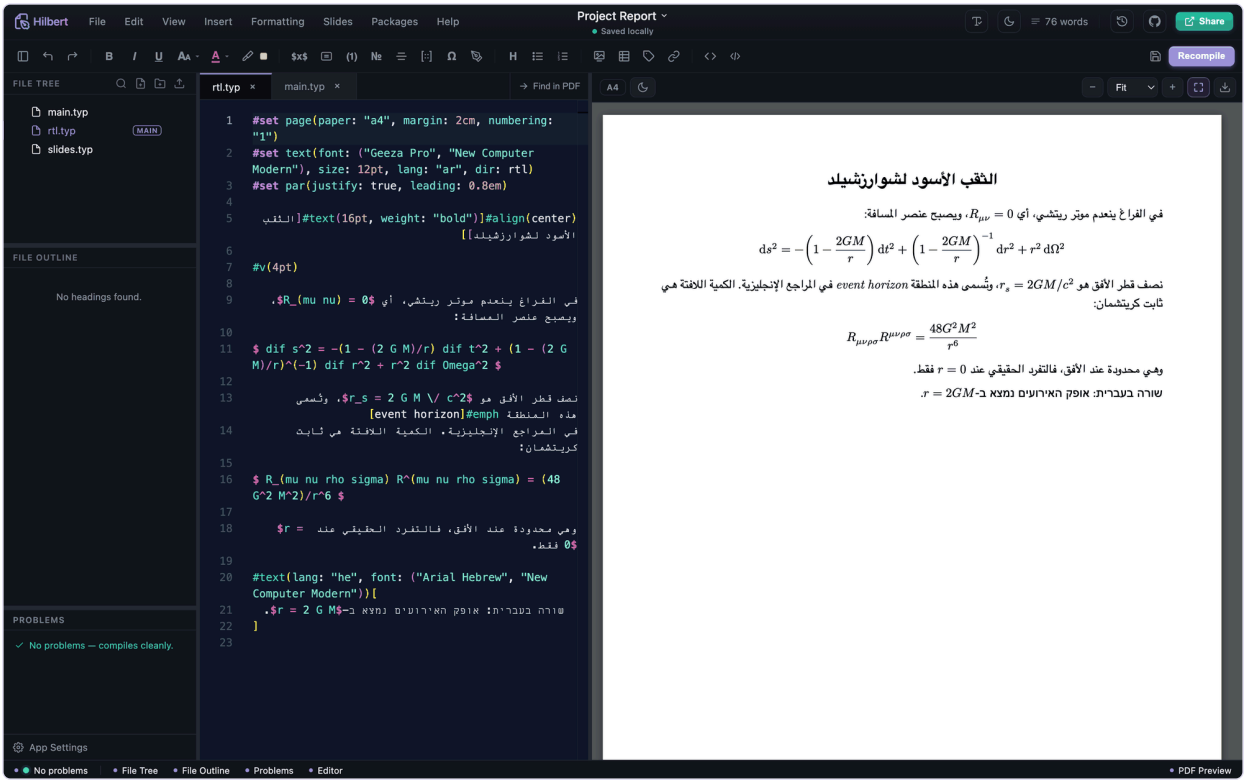


Figure 6: An Arabic document: prose runs right to left, the Typst code and the equations stay left to right, and the PDF joins the letters properly.

9.1 The document's own direction

Edit → Document Settings writes the `#set text(...)` rule at the top of the file:

1. Set **Language** to the ISO code — `ar`, `he`, `fa`, `ur`, `dv` and so on. The picker lists the common ones with their native names, but the field takes any code Typst accepts.
2. Set **Text direction** to **Right-to-left**, or leave it on **From the language**, which infers it: `ar`, `dv`, `fa`, `he`, `ks`, `pa`, `ps`, `sd`, `ug`, `ur` and `yi` are treated as right-to-left.
3. Choose a font that actually has the script.

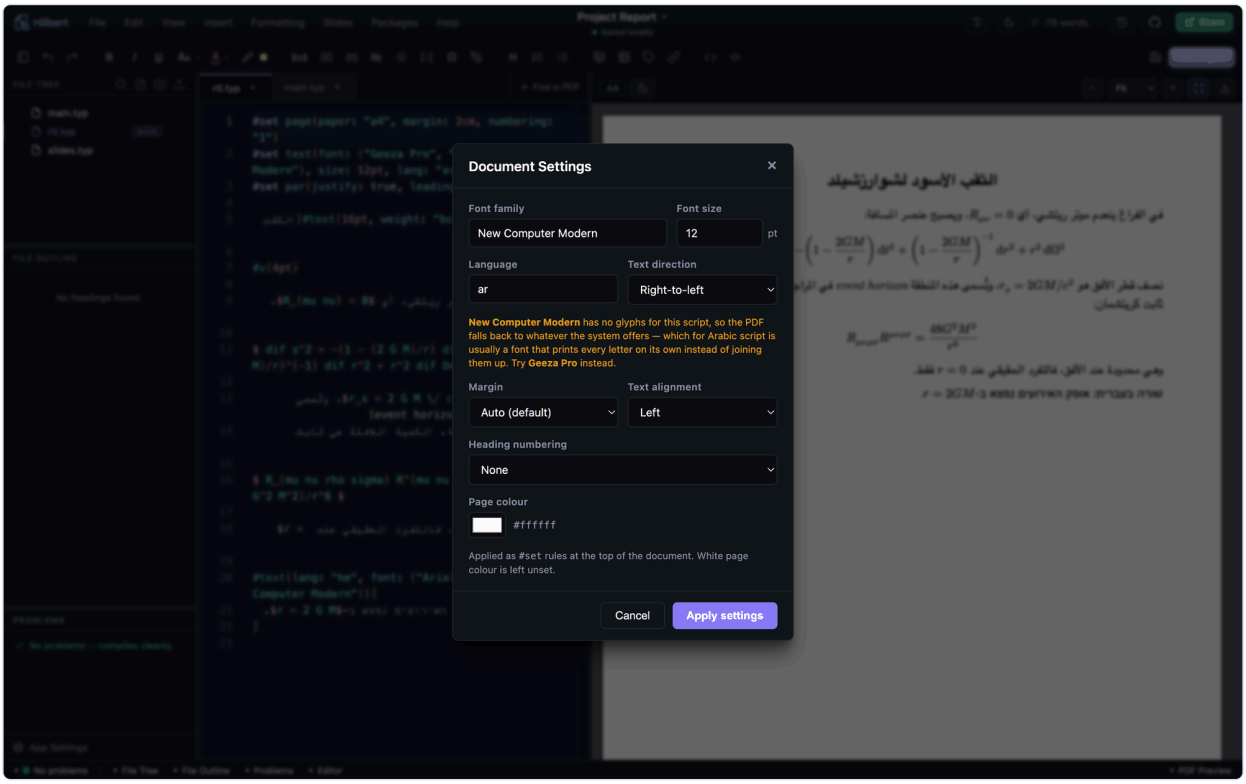


Figure 7: Document Settings on an Arabic document, with the font warning that stops the most common mistake.

That third step is where documents usually go wrong, so Hilbert checks it for you. If the document is right-to-left and the font named in the rule is a Latin one, the dialog warns in amber that the PDF will fall back to whatever the system offers — which for Arabic script is typically a font that prints every letter on its own instead of joining them up, producing text that is technically present and practically unreadable. It names an installed font that would work instead, or tells you none is installed and points at **File → Import Font**.

It also warns separately when the font named in the document is not installed on this machine at all, whatever the script.

9.2 How the editor lays out a mixed file

A Typst source file in Arabic is not an Arabic text file. The prose runs right to left, but `#set`, `#figure`, function names, labels and every equation are written left to right, and a line usually contains both.

Each line gets its own base direction, taken from the first strong character in it — rules P2 and P3 of the Unicode bidirectional algorithm. Crucially the scan is **syntax-aware**: it skips maths, raw blocks and code runs before looking for that first strong character, because a line like `#emph[שלום]` would otherwise be called left-to-right on the strength of the `e` in `emph`.

Three modes are available from the status bar and **App Settings**:

Mode	Behaviour
Automatic	Each line follows the script you typed it in. This is what you want nearly always.
Left-to-right	Always, even for Hebrew or Arabic.
Right-to-left	Always, even for English. The whole interface mirrors — the file tree, the outline indentation and the fold arrows included.

For the individual line the heuristic gets wrong, **Edit → Flip This Line's Text Direction** forces it one way, then the other, then back to automatic.

9.3 Fixing a phrase, not a line

When an English term inside an Arabic sentence lands in the wrong place — or a number, or a bracketed citation — the fix is an isolate, and the Insert menu has all three:

Command	What it wraps the selection in
Keep Selection Right-to-Left	RLI … PDI — an isolated run that reads right to left
Keep Selection Left-to-Right	LRI … PDI — an isolated run that reads left to right
Keep Selection Together	FSI … PDI — the run picks its own side from its first strong character

An isolate opens a run the surrounding text cannot reorder, which is why it fixes phrases that marks alone cannot. The older single-character marks are available too: RLM, LRM and the Arabic letter mark ALM, the last for the case where Arabic-Indic digits need to stay with their Arabic context.

9.4 Seeing the characters that move your text

All of those controls are invisible characters, and so are the legacy embedding and override codes that a file may already contain. Hilbert draws a hairline wherever one sits, and hovering it names the character and its code point — right-to-left mark (U+200F), pop directional isolate (U+2069), and so on. Ordinary control characters get the same treatment.

A file that reorders itself around something you cannot see is a file you cannot debug, which is the whole reason for showing them.

CHAPTER 10

Maths and physics

10.1 Symbols

A maths and physics symbol picker (⌘⇧P) backed by the `physica` package, and a draw-a-symbol pad (⌘⇧Y) that matches your sketch against the glyph shapes — entirely offline, with no service involved.

10.2 Ready-made physics

A Physics & Cosmology menu of compile-checked equations: bra-kets, commutators, the Dirac and Klein–Gordon equations, the QED Lagrangian, Einstein's field equations, Christoffel symbols, the FRW metric and the Friedmann equations. An equation gallery of fill-in templates sits alongside it.

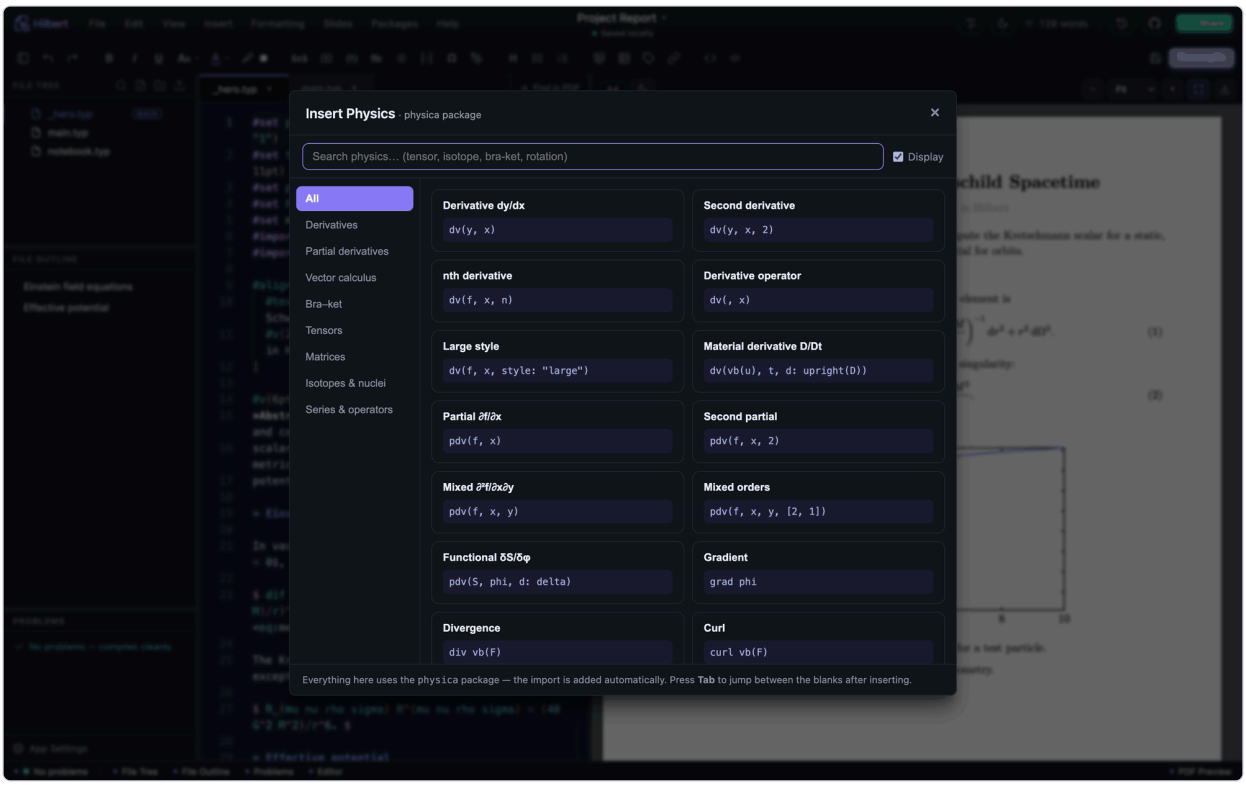


Figure 8: The physics equation menu.

CHAPTER 11

Plots and diagrams

11.1 Plot Studio

One tool for every plot, emitting `cetz` and `cetz-plot`:

- 2D functions — explicit, implicit and parametric;
- 2D data — line, scatter and bar;
- 3D `cetz` surfaces;
- launchers into the interactive 3D studio and the Python/matplotlib runner.

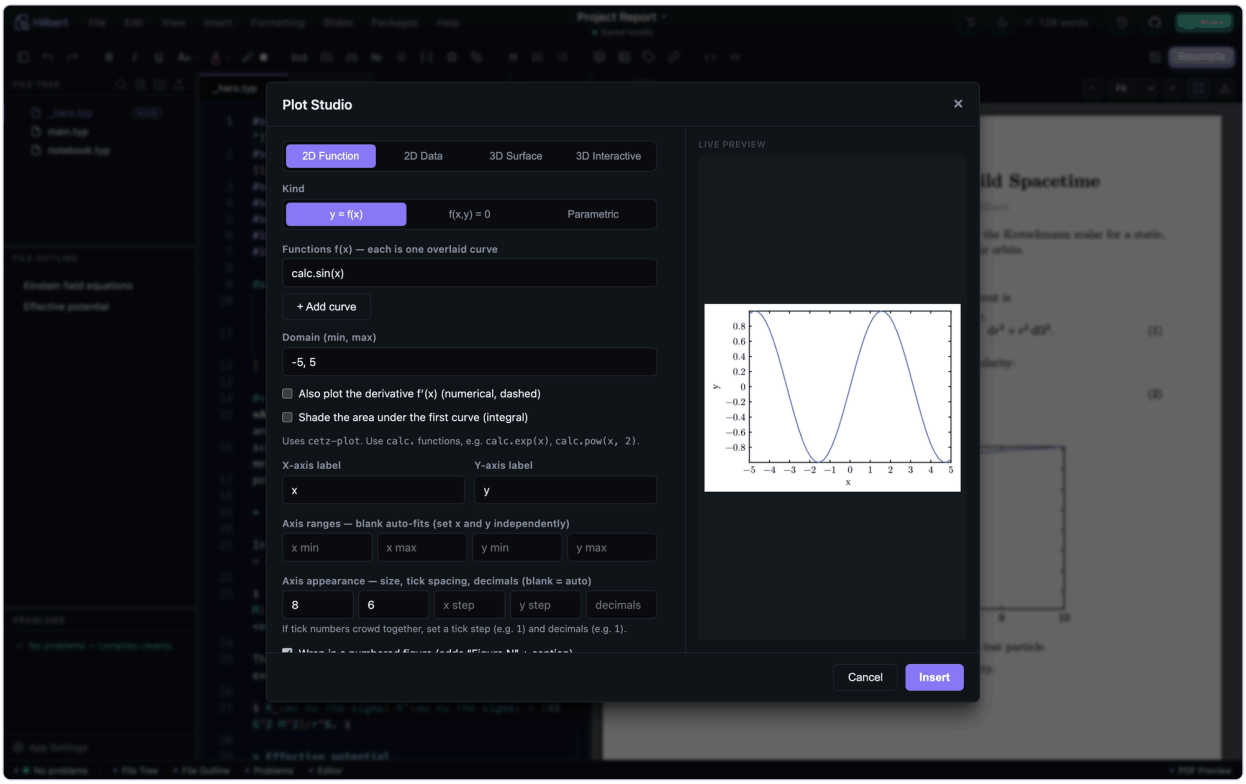


Figure 9: Plot Studio with a live preview of the curve.

11.2 3D Plot Studio

A surface you rotate by hand until it looks right, then insert exactly that view.

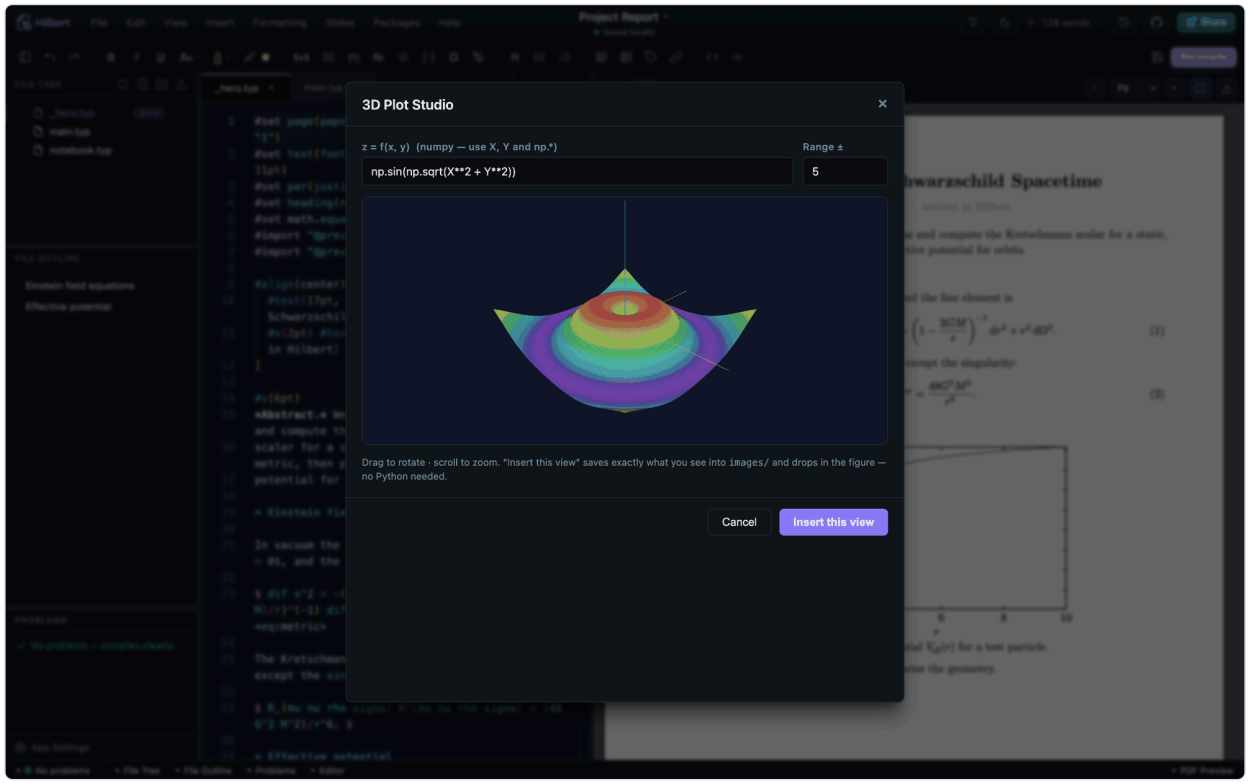


Figure 10: The interactive 3D studio.

11.3 cetz Canvas

A visual shape builder with thirteen primitives — circle, ellipse, rectangle, triangle, hexagon, line, arrow, arc, curve, grid, point, axes and label — a live preview, and per-shape position, size, rotation and colour. It will also plot a curve straight from a data file once you pick the X and Y columns.

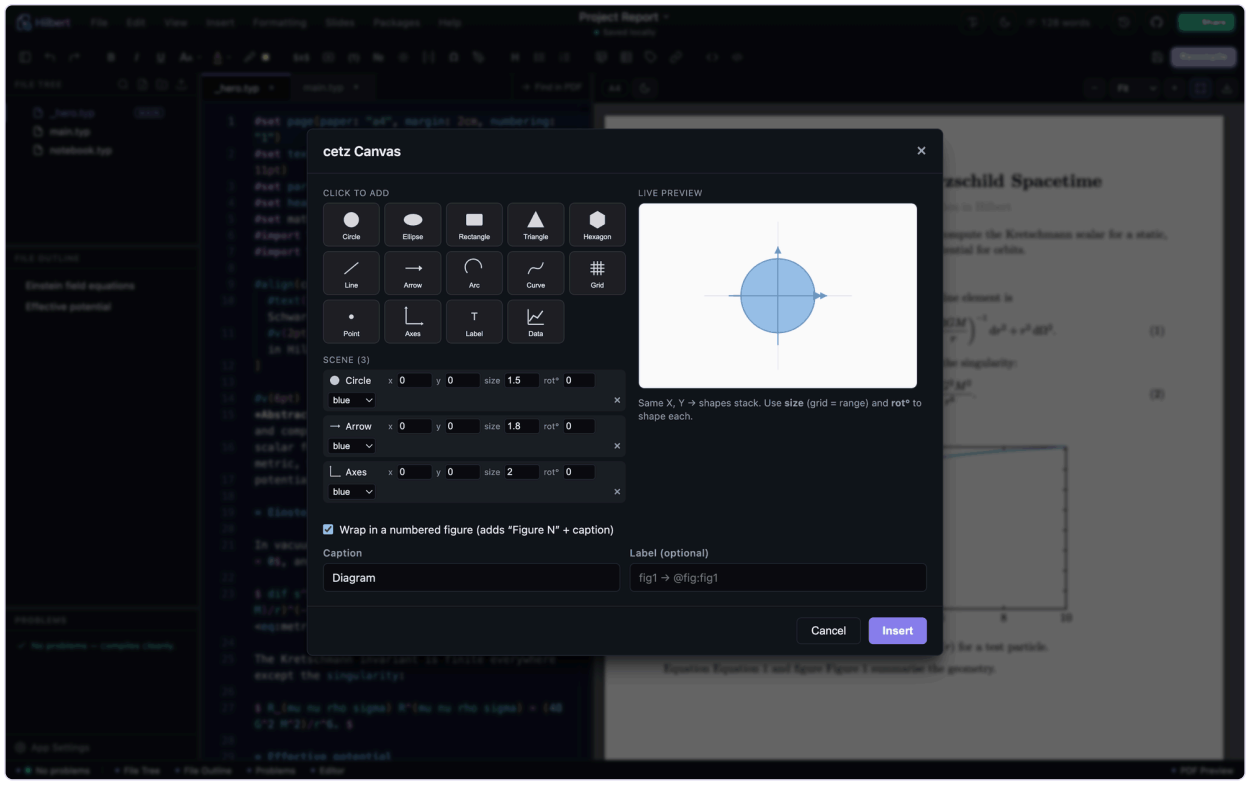


Figure 11: cetz Canvas: shapes, axes and labels with a live preview.

11.4 Diagrams

- **Commutative diagrams** come from a bundled offline copy of quiver, inserted as editable `fletcher`.
- **Feynman diagrams** (⌘⇧F) are drawn visually and come out as editable `cetz`.
- **Flow diagrams** use `fletcher`; **Flowchart → Code** (⌘⇧L) turns drawn logic into real `while`, `if` and `for` statements.

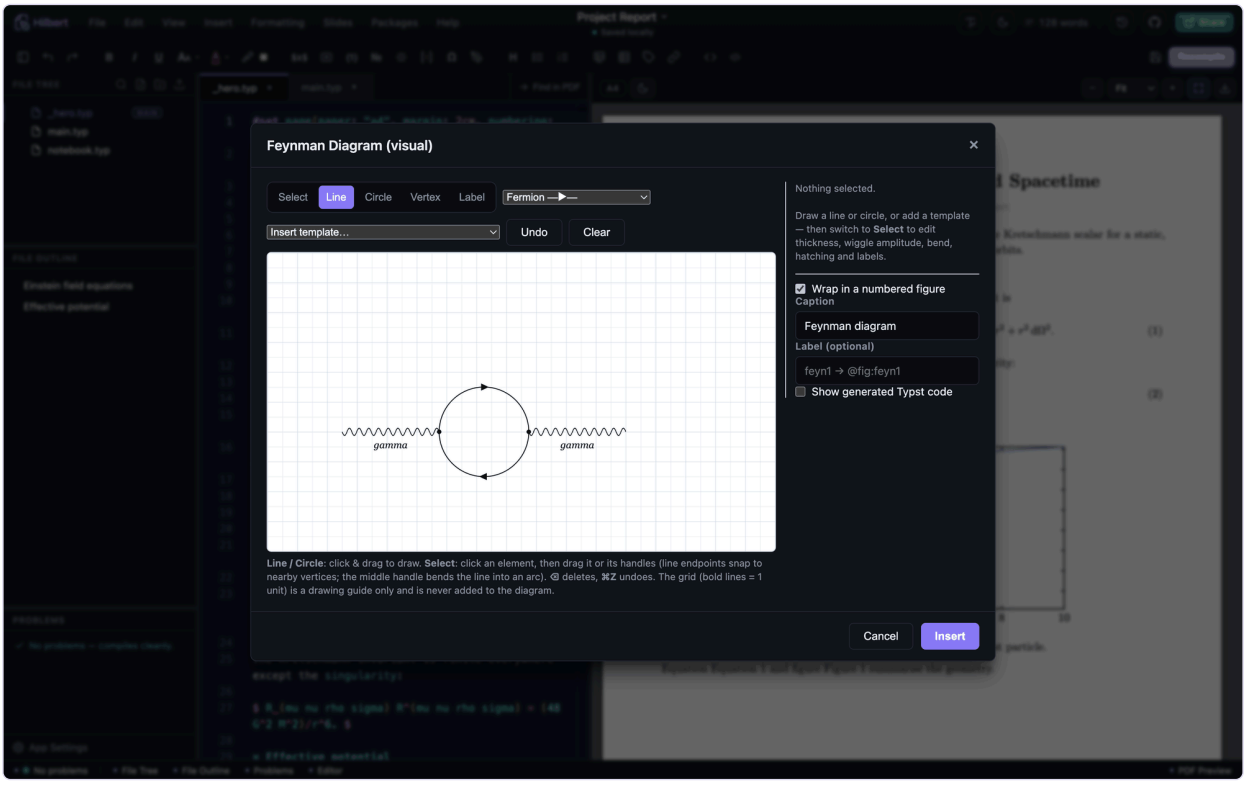


Figure 12: The Feynman diagram builder.

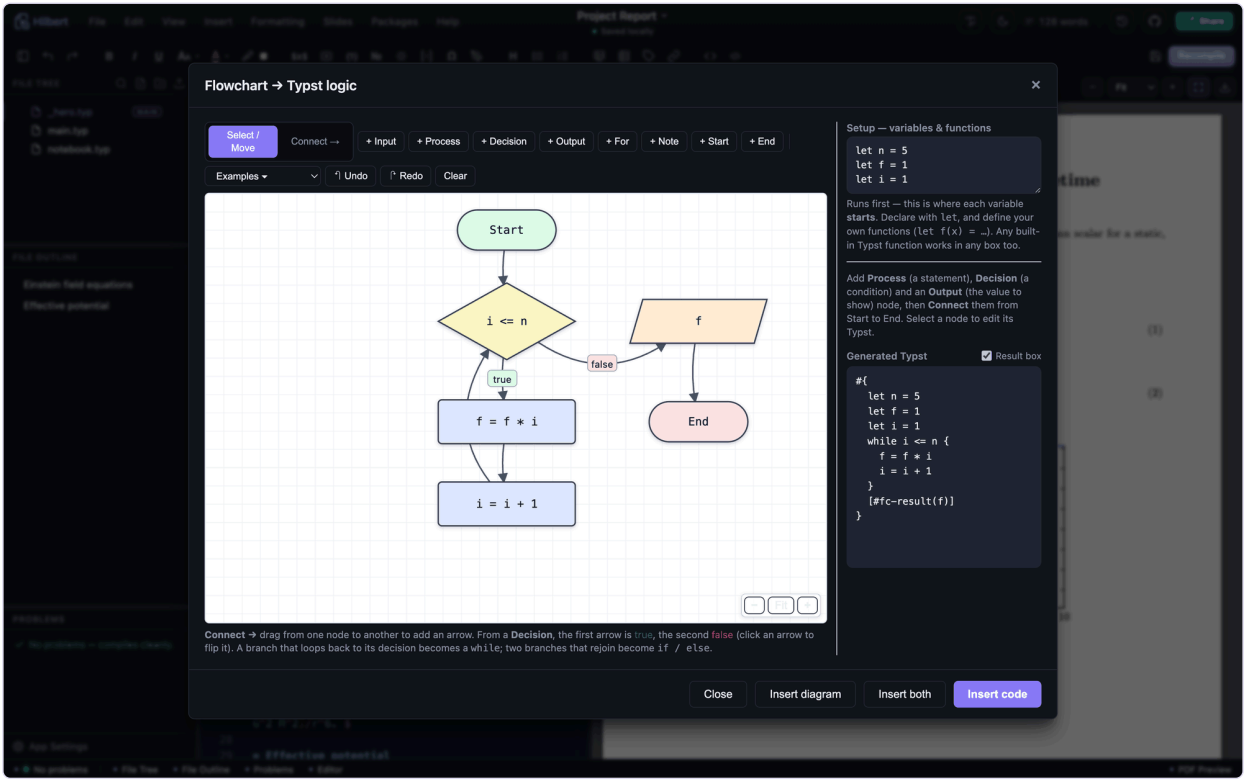


Figure 13: Flowchart to Code: the diagram and the code it generates.

CHAPTER 12

Maths that computes

12.1 Running a snippet

Run Python (⌘⇧K), **Run Julia** and **Run Wolfram** execute a snippet and insert the result as text output, as a generated figure, or — in *equation mode* — as a typeset equation. In equation mode you write plain maths like `diff(sin(x**2), x)` and get the typeset derivative back.

12.2 Run Notebook

Run Notebook executes every ```` ```python ```` and ```` ```julia ```` block in the document as a single session, so variables persist from one cell to the next. Output and plots land underneath each block, and the compiled PDF badges each block with its language logo.

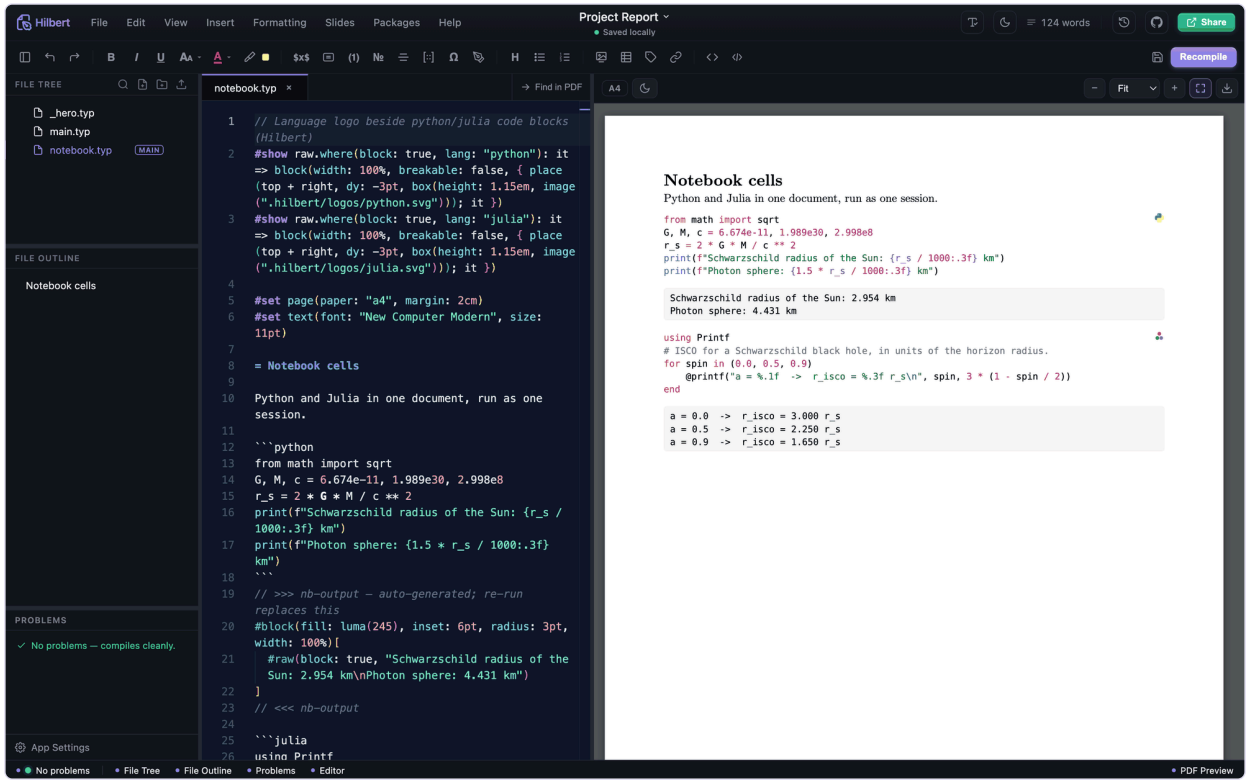


Figure 14: Python and Julia cells run as one session, output written back beneath each.

A run has guardrails whether or not a sandbox is available: a wall-clock timeout (`EXEC_TIMEOUT_MS`, 45 seconds by default), OS limits on file size and CPU, and output truncated at 8 MB, so a runaway cell cannot fill the disk or exhaust memory. Plots the run produces are moved out of the scratch directory into a visible `assets/` folder, because the document embeds them and they have to outlive the cleanup.

12.3 Compute on a selection

Highlight an expression and **Compute Selection** (⌘⇧U) will simplify, solve, differentiate, integrate or evaluate it with sympy, dropping the result back in as an equation.

12.4 The bundled physics examples

The runner ships with worked examples: General Relativity with xAct — Schwarzschild curvature through to the Ricci tensor and the Kretschmann scalar — Penrose diagrams, and Clebsch–Gordan and Wigner 3-j coefficients, as a rendered image or a typeset equation.

CHAPTER 13

References and bibliography

13.1 Labels and cross-references

Add a label to a heading, equation or figure (`= Intro <sec:intro>`), then type `@` and pick it from the completion list. The **Reference & Label Manager** lists every label and every `@reference` in the project, flagging the undefined, the duplicated and the unused.

13.2 Citations

Citations & Bibliography looks a paper up by DOI or arXiv id — or a URL — saves it into `refs.bib`, cites it with `@key`, and adds the bibliography section if the document does not have one yet. The lookup is the only part that needs a network connection; everything after it is local.

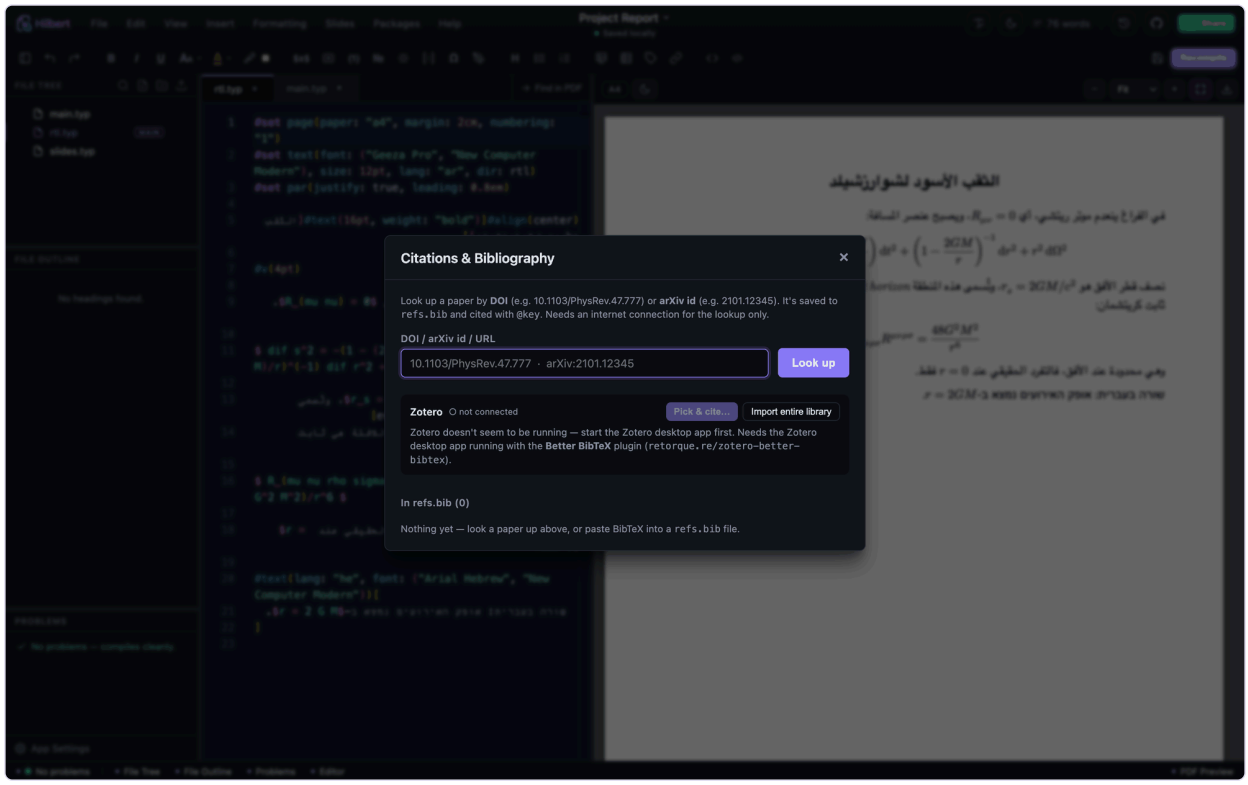


Figure 15: The citation manager. It also lists what is already in `refs.bib`, so you can cite a paper you saved earlier without leaving the dialog.

Zotero is supported directly: with the Zotero desktop application running and the Better BibTeX plugin installed, **Pick & cite** browses your library, and **Import entire library** brings the whole thing into `refs.bib` at once.

CHAPTER 14

Slides

Slide Studio builds 16:9 decks visually, and stores them as ordinary Typst source, so a deck you built last month reopens and edits rather than turning into generated output you dare not touch.

Open it with **Slides → Slide Studio**, or from ⌘K.

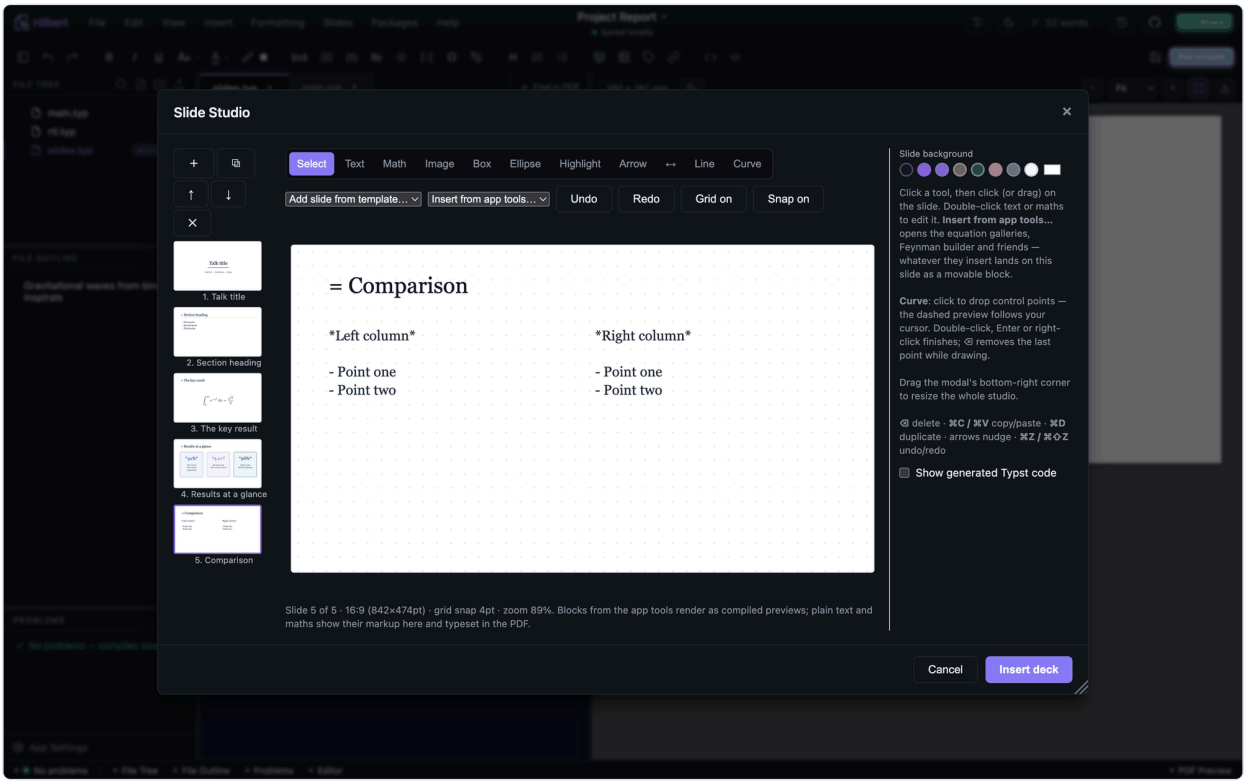


Figure 16: A five-slide deck. The rail on the left reorders by dragging; the panel on the right explains whichever tool is selected.

14.1 Building a deck

1. **Add slide from template** gives you a starting layout: Title slide, Heading + bullets, Two columns, Section divider (dark), Big equation, Agenda, Image + caption, Two images at the same height, Quote / key message, or Three key results.
2. Click a tool, then click or drag on the slide. The tools are Select, Text, Math, Image, Box, Ellipse, Highlight, Arrow, double-arrow, Line and Curve.
3. Double-click any text or maths block to edit it in place.
4. Reorder slides by dragging their thumbnails, or with the up and down buttons; duplicate and delete are next to them.

The Curve tool works by clicking control points, with a dashed preview following the cursor; double-click, Enter or right-click finishes the curve, and backspace removes the last point while you are still drawing.

Grid and snap are toggles, both on by default, with a 4 pt snap. The whole studio window resizes by dragging its bottom-right corner.

Shortcut	Action
⌫	Delete the selected block

⌘C / ⌘V	Copy and paste a block
⌘D	Duplicate
Arrow keys	Nudge
⌘Z / ⌘⇧Z	Undo and redo

14.2 Pulling in the rest of the application

Insert from app tools drops the output of another builder onto the current slide as a movable block: the Equation Gallery, the Physics Gallery, Matrix Studio, the Feynman builder, cetz Canvas, a commutative diagram, Plot Studio or the flowchart tool. Whatever they generate arrives as a block you can position like any other.

Blocks that come from those tools render as compiled previews inside the studio. Plain text and maths show their markup while you are editing and typeset properly in the PDF.

14.3 It is just Typst underneath

Tick **Show generated Typst code** and the source appears below the canvas.

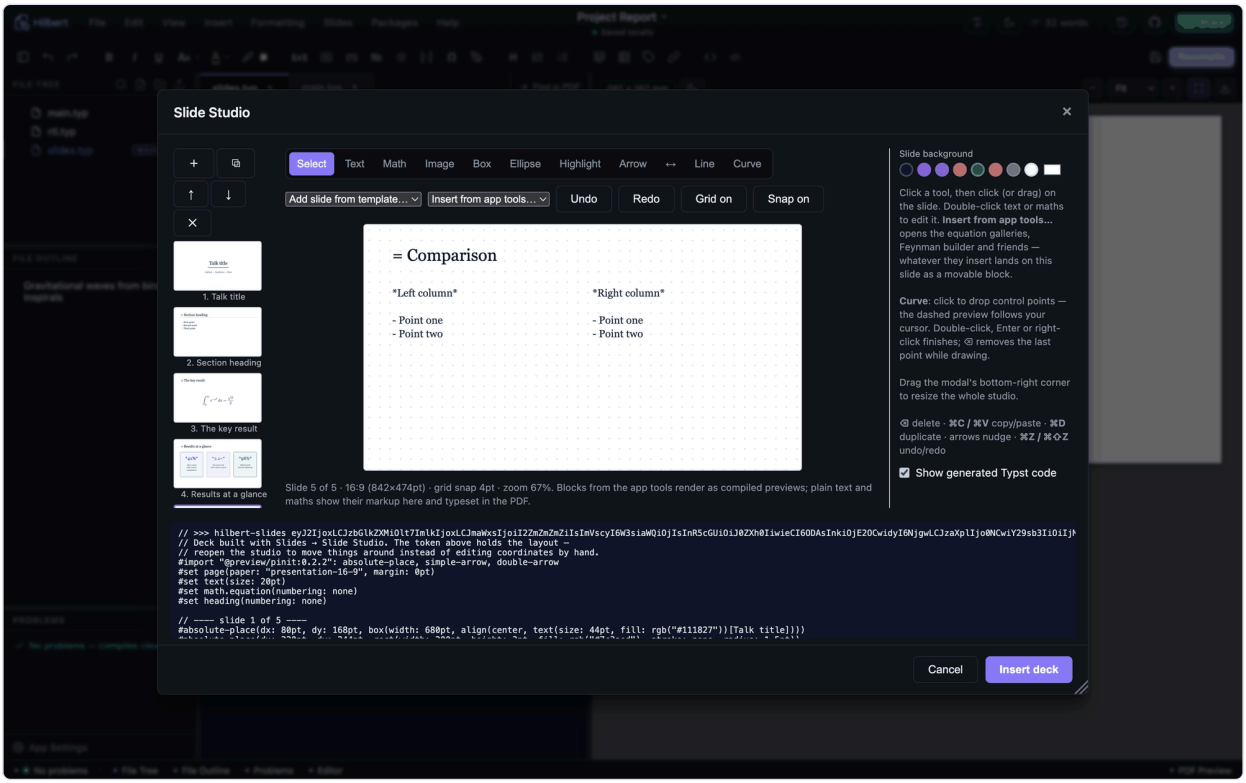


Figure 17: The deck's own source. The token on the first line holds the layout, so reopening the studio is lossless.

The first line is a comment holding the layout as a token, which is what lets the studio reopen a deck and give you the same movable blocks instead of a wall of coordinates. Everything under it is ordinary Typst — `#absolute-place` from `pinit`, a `presentation-16-9` page rule, your text — so the deck compiles with plain `typst compile` and survives without the studio if you ever want to edit it by hand.

Insert deck writes it into your document. The Slides menu also carries two `pinit` helpers for annotating slides: pin a highlight with an arrow note, and draw an arrow between two words.

Performance while dragging is flat in the length of the deck — see Section 7 for the numbers.

CHAPTER 15

Getting things in and out

15.1 Importing data

CSV, TSV and Excel files (`.xlsx` , `.xls` , `.ods`) come in through a preview, and you choose whether to insert them as a Typst table, as a plot with the columns you pick, or as a variable. JSON, YAML and TOML arrive with the matching Typst reader already wired up.

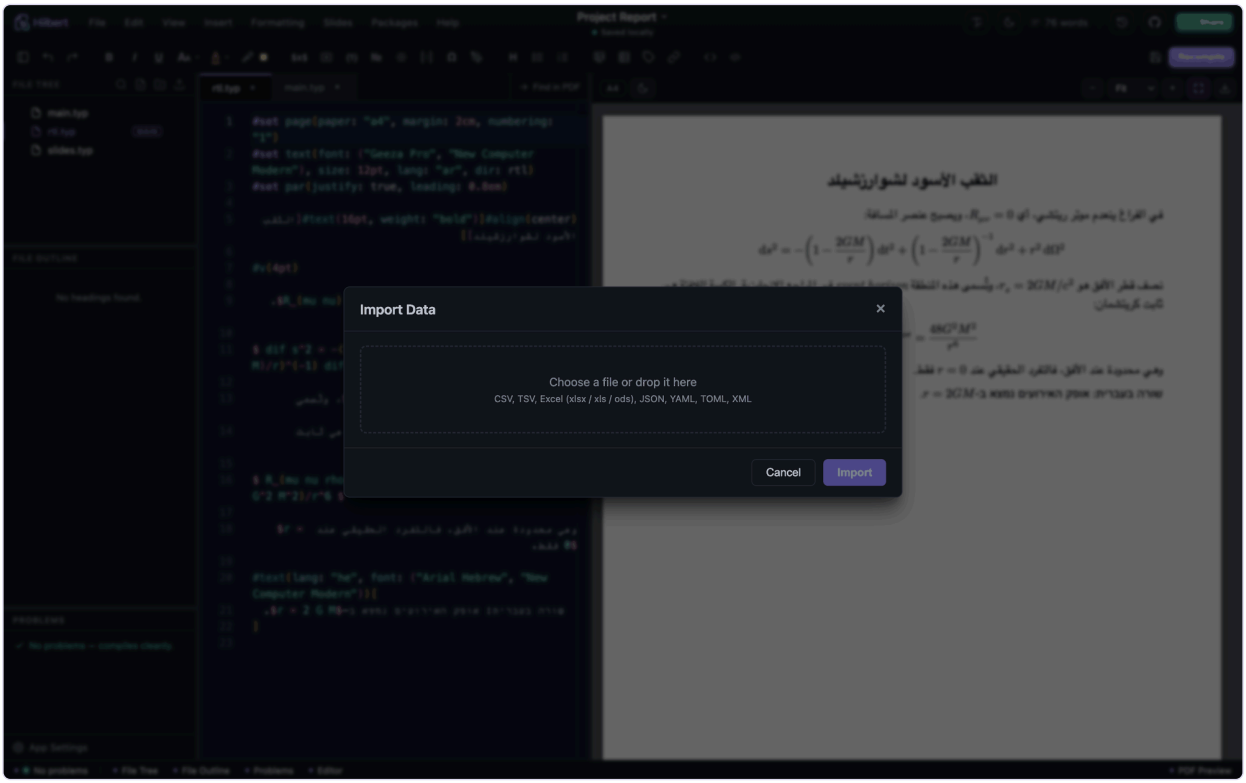


Figure 18: Import Data takes a file from the dialog or a drop.

15.2 Templates

Templates come from Typst Universe with a rendered preview, and six ship with the application for offline use — among them a two-column journal paper and a LaPreprint-style preprint with margin notes, ORCID links and a running footer.

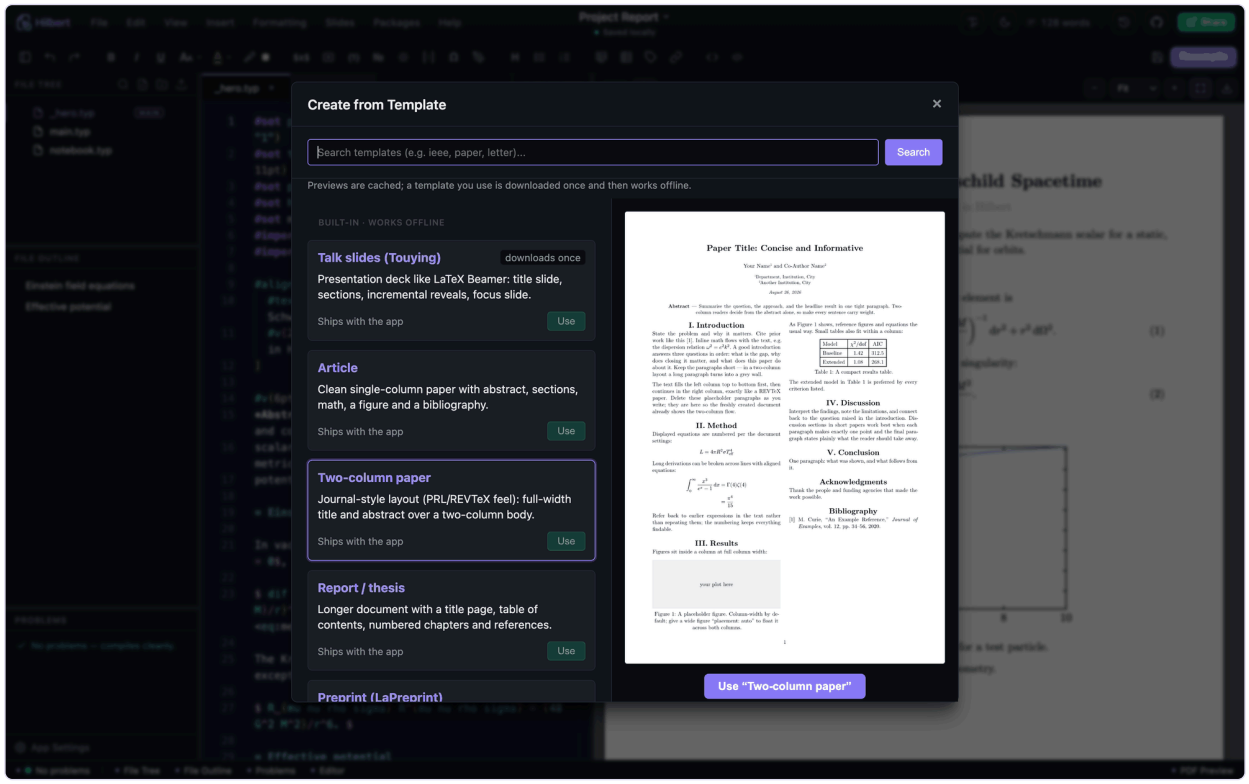


Figure 19: The template browser, with offline templates alongside Typst Universe.

15.3 Exporting

File → Save As / Export goes to PDF — with page ranges, PDF/A standards, tagging and pretty-printing — as well as PNG, SVG, HTML, plain `.typ`, or the whole project folder, through your system's own save dialog.

15.4 Packages

Packages → Install Typst Package searches, downloads and removes Typst packages. Bundled packages are cached locally, so documents compile with no network and no downloads.

15.5 Git and sync

Git support covers init, commit and push to GitHub. Beyond that there is sync to a local folder, to Google Drive, or to WebDAV (Nextcloud and ownCloud). Cloud credentials live only in your browser's local storage, and a GitHub token is used for the one push you asked for and never written into `.git/config`.

CHAPTER 16

Live collaboration

> Collaboration is **experimental**. It works, but expect rough edges — keep your own backup of anything important, and read `docs/COLLABORATION.md` in the repository before your first real session.

It is offline-first and account-free: no service in the middle, no sign-up, your machine talking to theirs. Everyone keeps a real copy of the project on their own disk and compiles it locally, so the preview each person sees is their own Typst build of the real document.

16.1 What actually travels

- Every text file — `.typ`, `.bib`, notes, code — merged live, character by character.
- Images, fonts, PDFs and other binary assets, sent separately and verified by content hash.
- Whiteboards, which sync when saved rather than stroke by stroke.
- Plots your code generates, so the other person sees the figure without running anything.
- File creations, renames and deletions.
- Everyone's cursor and selection, labelled with their display name.

One gap worth knowing: what travels is files, so a folder you create and leave empty does not appear for anyone else. It shows up the moment something is in it.

16.2 What you need

Hilbert on each person's computer, a network path between the machines, and an open text file on the host's side to start from. On the same Wi-Fi the network part is automatic; across a campus or the internet it needs one reachable address, which is what the table further down is about.

16.3 Hosting a session

1. Open the project, with a text file open in the editor.
2. `⌘K`, then **Share this project live** — or click Share and switch to the **Live** tab.
3. Accept the address Hilbert suggests, which on a home network is usually your LAN address, something like `ws://192.168.1.20:3020`. Or type a different one.
4. Enter a display name, so people can tell whose cursor is whose.
5. Start the session. The invitation lands on your clipboard; send it to the people you want in the room.

An invitation looks like this:

```
hilbert-collab://join?server=ws://192.168.1.20:3020&room=<random id>&key=<random key>
```

The room id and the key are generated fresh for every session.

> Anyone holding the invitation can join and read the entire project, because the key is in it. Send it over a channel you trust.

16.4 Joining

1. `⌘K`, then **Join a shared project**.
2. Paste the invitation.
3. Choose where the project should live. Hilbert offers a new folder, and — if you have joined before — the folders you used previously.

4. Enter your display name and join.

A badge in the header shows how many people are connected; click it for session details and the option to leave.

16.5 Rejoining later

Pick the **same folder** from the dropdown rather than making a new one, and Hilbert merges instead of starting over:

- Files the session also has are updated to the session's version, which is the merged state of everyone still connected.
- Files only you have are shared back up, so nothing you made while away is stranded.
- Files deleted while you were away are listed, and you are asked before anything on your disk is touched.

> The shared model carries no timestamps, so a file you edited locally **while disconnected** loses to the session's copy when you rejoin. If you did offline work you want to keep, copy it aside first.

16.6 Getting the network right

Situation	What to do
Everyone on one router	Host directly; the detected LAN address works as offered.
Across a campus or office	Firewalls usually block the direct route. Run a relay, or forward the port over SSH.
A dedicated relay	Run `hilbert --sync-server --port 3020` on a machine both sides can reach — a small server or a Raspberry Pi is plenty — and set its `ws://` or `wss://` address as the collaboration server.

16.7 What is encrypted, and what is exposed

Document updates, presence and cursors are encrypted end to end with the one-session key that travels inside the invitation, so a relay only ever handles ciphertext. The collaboration listener is a separate binary-only relay with bounded rooms, peers, frame size and traffic rate; it never exposes the workspace API, which stays bound to your own machine.

It starts on port 3020 when the application launches. `HILBERT_COLLAB_PORT` moves it, and `HILBERT_COLLAB=0` stops it starting at all, keeping Hilbert strictly local.

The host must stay online for a direct session. Saving, compiling and export keep working throughout, the files on disk stay yours, and leaving is deliberate: the key is not kept afterwards, so rejoining needs a fresh invitation.

16.8 When something is wrong

"The host was reached but no document synchronized." The relay is reachable but the session is not answering — usually the host left, or the invitation is from an older session. Ask for a fresh one.

Nothing happens when joining Check the address in the invitation is one your machine can reach — `ping` or `ssh` to it first. If the host is behind a firewall, port 3020 needs to be open inbound.

A file looks out of date Sessions reconcile every few seconds; give it a moment. If it persists, leave and rejoin the same folder to re-sync.

An image is missing from the preview The bytes travel separately from the text that references them, so a large asset can arrive a moment later. The preview recompiles itself once it lands.

Someone's whiteboard changes are not showing Whiteboards sync on save. Ask them to press `⌘S` in the whiteboard tab.

CHAPTER 17

Running it like Overleaf, in a browser

Everything so far has been the desktop application. The same binary can instead host the whole thing — editor, files, compiler, preview and collaboration — so that everyone works in a browser and the project lives on one machine. This is the Overleaf-shaped setup: a URL, a sign-in, a shared project, no installation for the people you share it with.

> This is different from the invitation-based collaboration in Section 16. There, everybody runs the desktop app and keeps their own copy of the project. Here the server's workspace is the authoritative copy and visitors have no local clone.

17.1 Trying it on your own machine first

Two minutes, and nothing to undo afterwards:

1. Pick a folder to serve — an existing project, or an empty directory.
2. Run the binary in hosted mode with a token of your choosing:

```
HILBERT_SERVER_TOKEN="replace-with-a-random-secret-of-at-least-32-characters" \
  hilbert --serve --bind 127.0.0.1 --port 3001 --workspace ~/Documents/paper
```

   `--serve` and `--hosted-server` are the same flag. On macOS the binary sits inside the application bundle:

```
/Applications/Hilbert.app/Contents/MacOS/hilbert
```

3. Open `http://127.0.0.1:3001` and sign in with that token.

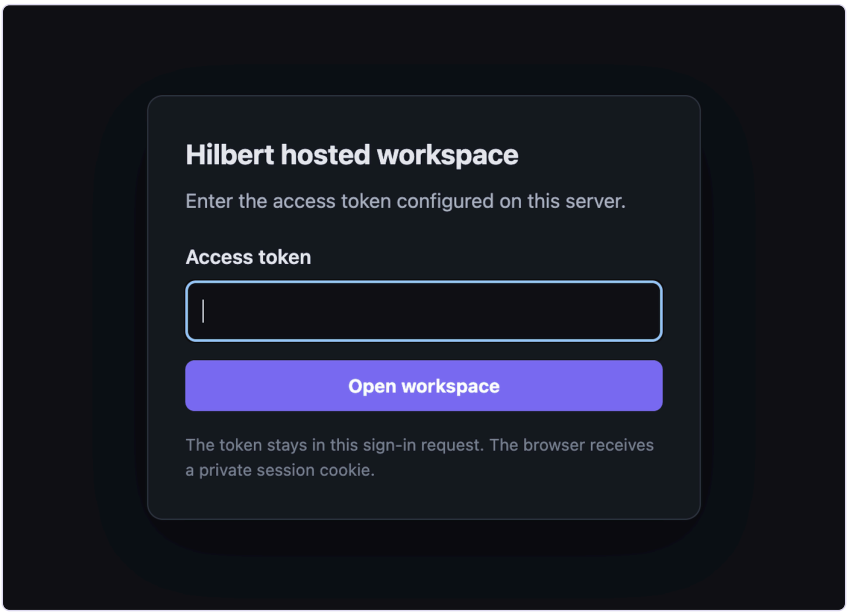


Figure 20: The sign-in page. The token is checked by this form only — the browser is then given a session cookie, so the secret never lands in a URL or in browser storage.

The editor that comes up is the one from the rest of this handbook: same menus, same builders, same preview.

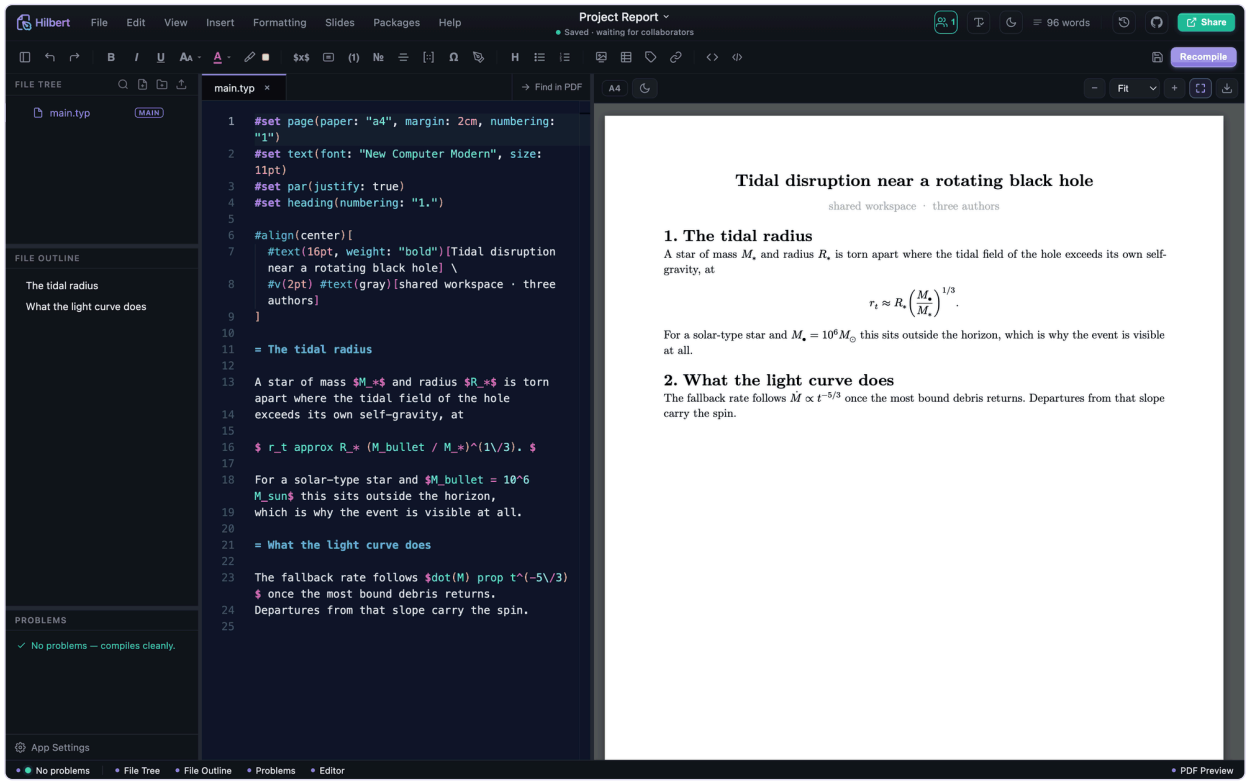


Figure 21: The hosted workspace in a browser tab, editing the server's project.

The first signed-in browser becomes the collaboration host and later browsers join it automatically. Edits, cursors, the preview, generated plots, whiteboards and saved files all travel through the same server.

17.2 Putting it on a server properly

A command typed into a terminal dies with the terminal. The repository's `deploy/` folder has what a real deployment needs: a systemd unit, an nginx server block, and a Caddyfile.

1. **Install bubblewrap first**, before enabling the service:

   ```
   sudo apt install bubblewrap        # or: sudo dnf install bubblewrap
   ```

   A hosted workspace refuses to run code it cannot confine. Without `bwrap` it still serves and compiles perfectly well; the Run buttons simply explain why they are unavailable.

2. **Create the account, the workspace and the environment file.** The four commands are listed in the header of `deploy/hilbert.service`. The service runs as an unprivileged `hilbert` user with its home under `/var/lib/hilbert`.

3. **Install and start the unit:**

   ```
   sudo install -m 644 deploy/hilbert.service /etc/systemd/system/hilbert.service
   sudo systemctl daemon-reload
   sudo systemctl enable --now hilbert
   journalctl -u hilbert -f
   ```

`/healthz` is an unauthenticated health endpoint, which is what a process monitor should poll. SIGINT and SIGTERM go through the normal shutdown path, so stopping the service leaves no orphaned compiler or language-server children behind.

> Three lines in that unit look like they should be tightened and must not be. `RestrictNamespaces` and `SystemCallFilter` are deliberately looser than a hardening guide would suggest, because bubblewrap

has to create namespaces and mount inside them to build the box user code runs in — tighten them and you get a service that looks well hardened, reports no sandbox available, and refuses to run anything. `MemoryDenyWriteExecute` is off because Julia compiles as it runs and needs writable-executable pages. And `ProtectHome=yes` hides `/home`, which also hides any interpreter installed there: install interpreters system-wide, or accept `ProtectHome=read-only` and that a cell can then read every home directory on the box.

17.3 HTTPS, and the two headers that matter

Hilbert binds to loopback and expects a reverse proxy to terminate TLS. Two headers in that proxy are load-bearing rather than boilerplate:

`Host` must pass through unchanged Hilbert checks each request's `Origin` against its `Host`, so a proxy that rewrites `Host` to the upstream name makes every browser request look cross-site. The symptom is unmistakable — the site loads, and then refuses everything you click with `403 Forbidden: cross-site request`.

`X-Forwarded-Proto` must say `https` it is how the application knows the visitor arrived over TLS, and it decides both whether the session cookie is marked `Secure` and whether the collaboration relay is offered as `wss://` or `ws://`.

The relay socket also needs a long read timeout, because it stays open for as long as someone has the document open and says nothing while nobody is typing. nginx's default 60 seconds closes it repeatedly and every editor reconnects in a loop. `deploy/nginx-hilbert.conf` has all of this. `deploy/Caddyfile` is shorter, because Caddy passes `Host` through and sets `X-Forwarded-Proto` by itself.

Encrypted collaboration needs a secure browser context. Over plain `http://<LAN address>` browsers withhold the Web Crypto API, so that address can edit and compile but cannot start the encrypted live channel. Use HTTPS even on a LAN, or the SSH tunnel below.

17.4 Without a certificate: the SSH tunnel

If the server is reachable by SSH but is not somewhere you can put a certificate, leave Hilbert on loopback and forward the port from each client:

```
ssh -N -L 3001:127.0.0.1:3001 user@server.example.edu
```

Then open `http://127.0.0.1:3001`. Browsers treat loopback as a secure context, so encryption works; SSH covers the network hop; and the Hilbert port is never exposed to anyone else on the network. Pick another local port on the left of `-L` if 3001 is taken.

17.5 The token, and signing people out

`HILBERT_SERVER_TOKEN` is the sign-in secret for every visitor. Under systemd it lives in `/etc/hilbert/hilbert.env` at mode 600, read as root before privileges are dropped. To rotate it:

```
printf 'HILBERT_SERVER_TOKEN=%s\n' "$(openssl rand -base64 48 | tr -d '\n')" \
  | sudo tee /etc/hilbert/hilbert.env >/dev/null
sudo chmod 600 /etc/hilbert/hilbert.env
sudo systemctl restart hilbert
```

Rotating signs everyone out, and does a little more than that: the hosted room and session keys are derived from the token and the workspace path, so a new token is a new hosted workspace identity. Open tabs can no longer reach the old room and must sign in again. Files on disk are untouched.

Signing in exchanges the token for a cookie, and that cookie is not the token — it is a short signed statement of when the session expires and which generation it belongs to, checked on every request. So:

- Sessions expire on the server rather than on the browser's honour. The default is 24 hours; `HILBERT_SESSION_HOURS` changes it, between 1 and 720.
- They survive a restart. A browser that was mid-edit when the service bounced picks up where it was instead of losing the draft behind a sign-in page.
- They can be ended without touching the token: `POST /auth/revoke-sessions` signs everyone out at once. Use that for a lost laptop, and rotate the token for a leaked token. `POST /auth/logout` clears only the calling browser.

17.6 Backups

The workspace directory is the authoritative copy of everything, and it is the only thing that needs backing up — that and `/etc/hilbert/hilbert.env`, which you can simply regenerate. Back it up the way you would any directory of source files: a `git` repository inside it works well and gives you history for free, with a nightly `rsync` or a filesystem snapshot to another machine for the case where this one dies.

Browser recovery drafts are not a backup. Hilbert does write each unsaved buffer to IndexedDB on the visitor's own device until the server confirms the save, and replays a draft whose server base is unchanged when the page next loads — if the server copy changed too, it keeps both and asks. That protects in-flight edits during an outage. It does not survive clearing site data, does not work through a different hostname or SSH port (a different origin), and cannot help at all while the server is down, because the browser cannot even load the interface.

17.7 What a browser tab cannot do

A few things belong to the machine the application is installed on rather than the one you are sitting at:

- Reveal in file manager, and opening a second native window.
- Cut, copy and paste use the browser's clipboard rather than the server's, so the first paste may ask permission — and Firefox restricts reading the clipboard from a page more tightly than Chrome does.
- Toolbar customisation is remembered per browser, so hiding a button changes nothing for anyone else.

Two ceilings protect the server's memory, and both apply only to what the browser is shown: a file preview stops at 64 MiB and a compiled PDF at 96 MiB. The file itself is untouched and the compile still succeeds; only the in-browser preview is refused, with a message saying so.

17.8 Running other people's code

Signed-in users can run the workspace's Python, Julia or Wolfram code when execution is enabled. Those runners have time and resource guardrails and, on Linux, a bubblewrap sandbox — but a sandbox is not a hardened boundary against a determined author. Set `ALLOW_CODE_EXECUTION=0` if collaborators should not run code at all, and use a dedicated OS account, a container or a VM when hosting documents you do not fully trust.

17.9 In a container instead

```
docker build -t hilbert-editor:latest .
docker run -d --name hilbert-editor \
  -p 127.0.0.1:8080:3001 \
  -e HILBERT_SERVER_TOKEN="a-random-secret-of-at-least-32-characters" \
  -v "$(pwd)/hilbert-workspace:/app/data" \
  hilbert-editor:latest
```

Open `http://localhost:8080` and sign in with that token. Leave `HILBERT_SERVER_TOKEN` out and the container prints a fresh one to `docker logs hilbert-editor` at every start. The mounted folder must be writable by uid 1000; if yours is not, add `--user "$(id -u):$(id -g)"`.

Bubblewrap cannot work inside an ordinary container, so the image sets `HILBERT_SANDBOX=off` and relies on the container boundary instead; `-e ALLOW_CODE_EXECUTION=0` turns code cells off entirely.

17.10 Building from source for hosted mode

From a source checkout there is no bundle to find the web application in, so you have to say where it is:

```
npm run build
cd src-tauri
cargo build --release
HILBERT_SERVER_TOKEN="replace-with-a-random-secret-of-at-least-32-characters" \
  TYPST_DIST=../dist \
  ./target/release/typst-editor --serve \
  --bind 127.0.0.1 --port 3001 --workspace /srv/hilbert/project
```

Use `--bind 0.0.0.0` only when other machines genuinely need to reach the port directly.

CHAPTER 18

Running from source

Everything lives in one repository: the React/Monaco frontend at the root, the Rust backend under `src-tauri/`. You need Node.js 18 or newer and a stable Rust toolchain. The first run compiles the backend, which takes a while.

```
git clone https://github.com/aburousan/hilbert-editor.git
cd hilbert-editor
bash scripts/setup.sh   # Typst + Python deps + npm install (macOS/Linux)
npm run dev             # UI on http://localhost:5173, backend on 127.0.0.1:3001
```

`npm run dev` serves the interface with Vite and hot reload and starts the Rust backend headless on port 3001. For the real desktop application, `npm run desktop` builds the frontend and opens the native window — that is also what a release build ships.

On Windows: `winget install --exact --id Typst.Typst`, then `npm install`, then `npm run dev`.

CHAPTER 19

Keyboard shortcuts

On Windows and Linux, read `⌘` as `Ctrl`.

Shortcut	Action
`⌘K`	Command palette — every action in the application
`⌘S`	Save and recompile now
`⌘B` / `⌘I`	Bold / italic
`⌘/`	Comment or uncomment the line or selection
`⌘E`	Inline equation
`⌘⇧E`	Numbered equation
`⌘⇧G`	Equation templates
`⌘⇧N`	Toggle numbering at the cursor
`⌘⇧M`	Matrix Studio
`⌘⇧P`	Maths and physics symbols
`⌘⇧Y`	Draw a symbol
`⌘⇧U`	Compute selection — simplify, solve, integrate
`⌘⇧H`	Full-width horizontal line
`⌘⇧B`	Code block
`⌘⇧F`	Feynman diagram
`⌘⇧L`	Flowchart to code
`⌘⇧K`	Run Python
`F2`	Rename symbol across the file

CHAPTER 20

Everything in the command palette

⌘K searches all of these by name. Grouped as the palette groups them.

File
New File · New Window · Open File · Open Folder as Workspace · Import Folder into Project · Import Font · New from Template · Save · Save As / Export · Sync / Share — Drive, WebDAV

Edit and code
Undo · Redo · Find · Find & Replace · Go to Definition · Find References · Rename Symbol · Quick Fix / Code Actions · Format Typst Document · Comment / Uncomment Lines · Toggle Numbering at cursor · Toggle Equation Numbering for all · Customize Toolbar · Document Settings · Flip This Line's Text Direction

View
Toggle Sidebar · Show Everything · Next Theme · Version History · Recompile Document · Preview HTML — experimental · App Settings — interpreters, git, cloud

Insert
Title Block · Author · Institute · Abstract · Heading · Theorem / Proof / Lemma · Keep Selection Right-to-Left · Keep Selection Left-to-Right · Keep Selection Together

Maths
Inline Equation · Block Equation · Multiline / Aligned Equation · Numbered Equation · Equation Templates · Insert Physics — physica · Matrix Studio · Matrix with augmentation lines · Conditional / Piecewise · Over / Under Brace · Cancel / Strike Term · Maths & Physics Symbols · Draw a Symbol · Compute Selection — simplify / solve

Lists and text
Bullet List · Numbered List · Nested List · Term / Definition List · Callout / Admonition box · Block Quote · Footnote · Margin / Side Note · Horizontal Line

Figures
Figure · Image · Whiteboard / Sketch — Excalidraw · Table · Import Data — CSV / Excel / JSON · Code Block · Subfigures side by side

Plots and diagrams
Plot Studio — 2D, data, 3D, Python · cetz Canvas — shapes & grid · Commutative Diagram — quiver · Feynman Diagram · Flow diagram — fletcher

Slides
Slide Studio — drag & drop deck builder · Pin highlight + arrow note · Pin arrow between two words

Compute
Flowchart → Code · Run Notebook — all code cells · Run Python · Run Julia · Run Wolfram

References
Link — web · Cross-reference — internal · Label · Reference & Label Manager · Citations & Bibliography — DOI / arXiv / Zotero

Formatting
Bold · Italic · Underline · Superscript · Subscript · Text Colour · Highlight / Background Colour · Page Setup · Box Selection · Font Size · Align Content · Rotate · Small Caps · Strikethrough · Letter Spacing

Collaborate
Share this project live · Join a shared project · Set optional collaboration server · Leave the shared project

Packages and help
Install Typst Package · Features & Help · Copy Diagnostics

CHAPTER 21

Environment variables

Set these in the environment Hilbert is launched from. Everything here has a working default; you need none of them for ordinary desktop use.

Variable	Default	Purpose
`ALLOW_CODE_EXECUTION`	`1`	`0` disables all code execution.
`EXEC_TIMEOUT_MS`	`45000`	Per-run wall-clock limit.
`HILBERT_SANDBOX`	`auto`	`auto` runs code unconfined where no sandbox exists; `require` refuses to run it; `off` never confines. Hosted mode defaults to `require`.
`HILBERT_SANDBOX_NET`	`0`	`1` keeps the sandbox but gives the code its network back.
`HILBERT_CODE_SCREEN`	`auto`	`always` keeps the source pattern screen on under a sandbox; `off` never screens.
`HILBERT_SERVER_TOKEN`	none	Required browser sign-in secret for `--hosted-server`, 32 characters or more.
`HILBERT_SESSION_HOURS`	`24`	How long a hosted browser session lasts, 1–720.
`HILBERT_PUBLIC_HOST`	none	Hosted only: the hostname this server is published as. Requests arriving under another name are refused.
`HILBERT_API_TOKEN`	generated	Optional API-token override, 32 characters or more.
`HILBERT_COLLAB`	`1`	`0` does not start the collaboration listener at all.
`HILBERT_COLLAB_PORT`	`3020`	Moves the collaboration listener off its default port.

CHAPTER 22

Where Hilbert keeps things

Settings, the last session and the activity log sit together in one folder:

Platform	Folder
macOS	`~/Library/Application Support/hilbert/`
Linux	`~/.config/hilbert/`
Windows	`%APPDATA%\hilbert\`

`settings.json` holds your preferences, `session.json` the project and open files you left behind, and `hilbert.log` the last few thousand lines of what the engine has been doing. Deleting any of them is safe — you get the defaults back.

Your documents live in `~/Documents/Hilbert` unless you open a folder elsewhere. Each workspace keeps scratch files in a hidden `.hilbert/` directory, which is also safe to delete.

CHAPTER 23

The security model

The backend is built for local, single-user use.

- It binds to `127.0.0.1` only, and CORS is limited to `localhost` and `127.0.0.1`. Requests carrying a foreign `Origin` or `Host` header are rejected, so a website you happen to have open cannot reach it.
- Every API request additionally needs a random bearer token minted at launch and handed only to the application's own window, so other local processes cannot drive the backend either. Scripted use sets `HILBERT_API_TOKEN` and sends `Authorization: Bearer <token>` with each request.
- File access is confined to the workspace, and path traversal is rejected.
- The collaboration listener is a separate binary-only relay with bounded rooms, peers, frame size and traffic rate. It receives only ciphertext.
- Code execution is time-limited, runs in a scratch directory under `.hilbert/run/` with OS resource limits on file size and CPU, and has its output capped — on top of the sandbox described in Section 4.

> Do not expose port 3001 to a network. The hosted server is the supported way to put Hilbert on one, and it has its own token and host checks.

CHAPTER 24

Troubleshooting

24.1 macOS says the app is damaged, or will not open

Gatekeeper quarantine, or a signature broken by renaming the `.app`. Run the two commands in Section 2.2, one per line.

24.2 The window is blank, or it reports that the local engine would not start

Something else is using port 3001. Quit it and reopen Hilbert.

24.3 It opens, but nothing compiles

The Typst CLI is missing or not on the `PATH`. Confirm `typst --version` works in a terminal, then check **App Settings → General**.

24.4 Tinymist works in a terminal, but the app says it is not installed

A running program keeps the environment it started with, and the desktop hands every application the environment it captured at login. A tool you installed afterwards — or one added to `PATH` by your shell's startup file, which the desktop never reads — is invisible until you log out and back in.

The fix, in order of effort:

1. Log out and back in, or reboot. This alone solves most cases.
2. On macOS, install into a location already on the system `PATH`, such as `/usr/local/bin` or Homebrew's prefix.
3. On Windows, set the variable as a *user-level* environment variable rather than in a shell session — a shell-set variable reaches only that shell, while a user-level one reaches the application and every process it spawns.
4. Send the output of **Help → Copy Diagnostics**, which includes the exact `PATH` that was searched.

The same reasoning applies to Python environments and conda that do not appear in **App Settings → Interpreters**.

24.5 It sits on "Compiling..." and will not finish

Past a few seconds the status bar says *still waiting on Typst* saving keeps working meanwhile. Recompile to start over. **Help → Copy Diagnostics** records every line `typst watch` emitted.

24.6 A template fails with an error inside `@preview/…`

A package compatibility problem, not an editor one: some Typst Universe templates pull in helper packages written for an older Typst. Your own document is fine.

24.7 A package will not download behind a corporate proxy

Typst itself fetches packages, and where a proxy intercepts TLS it needs to be told which certificate to trust. Typst reads `TYPST_CERT`; because Hilbert's child processes inherit the application's environment, setting it as a user-level variable (Windows) or exporting it before launching Hilbert (macOS, Linux) reaches the compiler.

24.8 `npm run dev` prints the concurrently line and stops

Run a full `npm install`, not `npm install --production`.

24.9 Before filing a bug

Run **Help → Copy Diagnostics**. It puts the activity log on the clipboard together with the Typst and tinymist it found and the `PATH` it searched, and the bug report form on GitHub has a box waiting for it. On Windows especially, a windowed application has no console to print to, so without that text a report can only describe the symptom.

Note that the diagnostics name the paths of the documents you had open — trim anything you would rather not post publicly.