Introducing zzvim-R

Ronald G. Thomas

2026-08-28

A single-file plugin that grew a workspace around itself: the same terminal pipe from the original prototype, now carrying plots, chunk navigation, and a Docker-aware R session.

Introduction

A little over a year ago I wrote about a 150-line Vim plugin, `rgt-R.vim`, built to understand Vim's terminal API well enough to send R code from an editing buffer to a running R session (see Writing a Simple Vim Plugin for REPL Interaction). That plugin has since become `zzvim-R`: a publicly released, actively maintained project at rgt47/zzvim-R, now closer to 3,800 lines, with a themis-based test suite, CI, and a `doc/` help file. This post is not a rewrite of the README. It is an account of what changed, why, and where the plugin sits today relative to the alternatives.

Motivations

- The original plugin only ever looked at `term_list()[0]`, meaning one terminal at a time; working across more than one R project in the same Vim session made that untenable.
- Chunk navigation relied on raw searches for ```` ```{ ```` fences, which broke on nested or malformed chunks and offered no way to jump straight to execution.

- None of the earlier design accounted for R running inside a Docker container, which is how most of my own R projects (zzcollab-scaffolded compendiums) actually run.

Objectives

1. Describe what changed between the 150-line prototype and the current plugin, and why each change happened.
2. Give a working definition of zzvim-R and place it honestly against the alternatives (R.nvim, ESS, RStudio, vim-slime).
3. Point readers toward the plugin's own documentation for installation and full mapping reference, rather than duplicating it here.

A single-purpose instrument that grew a small panel of readings around itself.

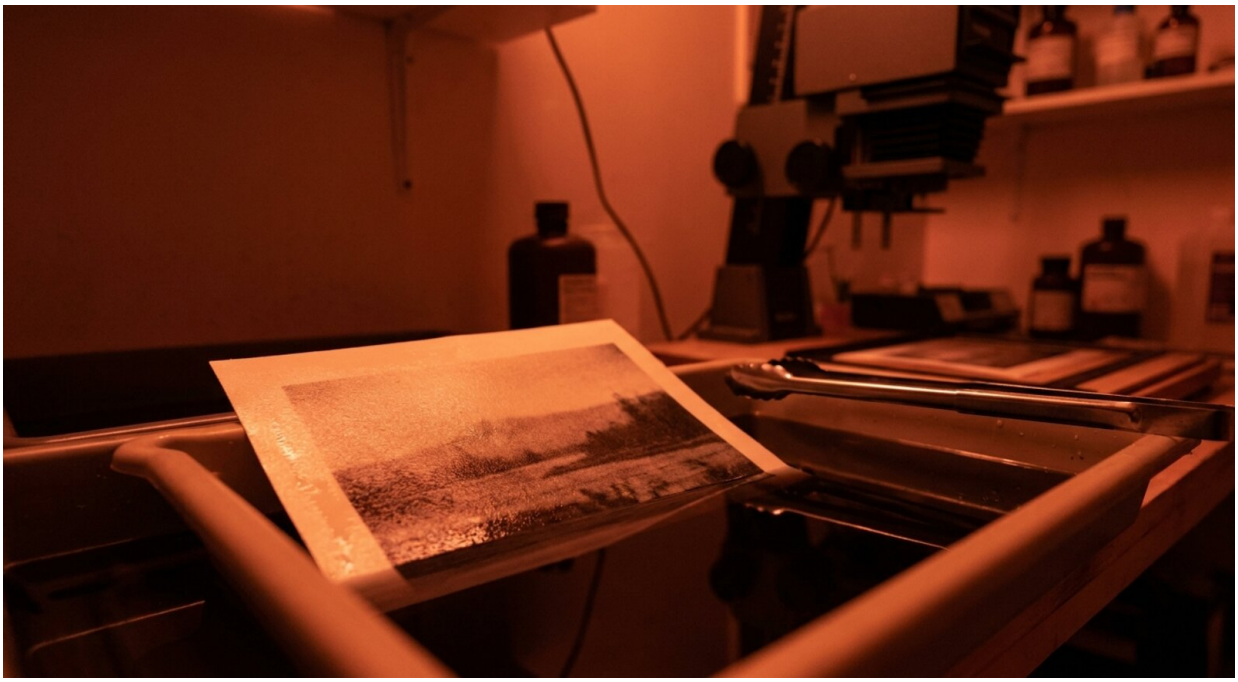
What is zzvim-R?

zzvim-R is a single VimScript implementation that targets both Vim 8+ and Neovim, with no Lua or external runtime dependency beyond R itself. It is closest in spirit to vim-slime: a terminal-centric REPL bridge rather than a client-server integration like R.nvim or ESS. What sets it apart from vim-slime is R-specific intelligence built directly into the plugin. It parses the buffer well enough to recognize a function body, a pipe chain, or a control block, and submits the whole unit on `<CR>` rather than a single line. It does not attempt to replace RStudio's viewer, R.nvim's object browser, or a GUI debugger. It is useful where the editing surface is already Vim or Neovim and staying there outweighs the convenience of a heavier IDE.

From Line Submission to a Workspace

The original `SubmitLine` function did one thing: copy the current line, append a newline, send it to the first terminal in `term_list()`. zzvim-R's `<CR>` mapping still sends code to a terminal, but it first inspects the buffer around the cursor to decide what 'the current expression' actually is, so pressing Enter inside a multi-line `dplyr` pipe or an unfinished `function()` body sends the whole thing, not one fragment of it. Terminal sessions are now tracked per buffer instead of taking whichever terminal happens to be first in the list, which is what makes working across several R projects in one Vim session practical.

Two capabilities did not exist at all in the prototype. The workspace HUD (`<LocalLeader>0`) opens a tabbed dashboard over memory usage, loaded data frames, and attached packages, sourced directly from the live R session rather than typed by hand each time. Inline plots go further still: zzvim-R renders a PDF master and a PNG preview and displays the PNG directly in the terminal pane, for the terminals that support it (Kitty, Ghostty, WezTerm, iTerm2), which closes a real gap the original design left completely unaddressed: it had no plot story at all.



A plot resolving into view in the terminal pane, the way a print resolves in a developing tray.

The Docker integration answers the motivation above directly: zzvim-R detects a zzcollab-scaffolded project (via the `.zzcollab` marker file) and launches R inside the container through `make r`, using the project's `renv` environment, rather than assuming a bare `R` on the host `$PATH`.

A self-contained unit that carries its own environment wherever it runs.

How It Compares

zzvim-R does not try to be everything. Against R.nvim (the Neovim-only, actively maintained successor to the archived Nvim-R), it gives up an object browser, richer completion, and inline help lookup. In exchange, it works identically in Vim and Neovim and ships the plot pipeline and Docker awareness R.nvim does not have. Against RStudio and VS Code, it gives up a graphical debugger, viewer pane, and polish in exchange for editing speed, a smaller footprint, and working the same way over SSH as it does locally. Against vim-slime, its closest relative, it trades generality across languages for R-specific pattern detection, chunk navigation, and the HUD. None of these are close calls in every direction; which one is right depends on whether the editing surface is already Vim, and how much the convenience features are worth relative to staying there.

Wrapping Up

What started as an afternoon spent reading Vim's terminal-API documentation turned into something used daily, then into something worth releasing. The lesson that carried through the whole process is one already stated in the original post: minimal tooling, understood completely, is easier to extend than someone else's feature-rich plugin. zzvim-R is the same idea after a year of extension; it is no longer minimal by line count, but every addition traces back to a limitation that showed up in actual use, not a feature added for its own sake.

Readers who want to try it should start with `docs/quickstart.md` in the plugin repository, a ten-step tour, rather than this post.

See Also

Related posts:

- Writing a Simple Vim Plugin for REPL Interaction: the origin story and the complete code of the original prototype
- Setting Up Neovim as a Data Science IDE: editor configuration, where zzvim-R appears as a lighter alternative to R.nvim

Key resources:

- zzvim-R: the plugin's GitHub repository
- zzvim-R comparison guide: the full comparison against R.nvim, ESS, RStudio, VS Code, and vim-slime
- vim-slime: the generic REPL bridge zzvim-R is closest to in design

Related posts in this cluster

This post is part of the *R Package Development and Testing* series. Recommended reading order:

1. Updating an R Package: A Complete Development Workflow
2. Writing a Simple Vim Plugin for REPL Interaction
3. **Introducing zzvim-R** (this post)
4. Testing Data Analysis Workflows in R
5. From testthat to tinytest: Converting an R Package Test Suite
6. Code Review for AI-Assisted R Package Development
7. Exemplar Code Review: The zzobj2fig Package