Colbyn Wadman

I specialize in building custom tools and utilities — but I've tackled a wide range of problems beyond that

APPS TOOLS AUTOMATION DATA/AI PIPELINES SOLO BUILDER TECHNICALLY FLUENT WRITER

hello@colbyn.com colbyn.com 801-367-4487

FACTS

- Top percentile OS contributor on GitHub.†
- Most popular authored OS tool has `688` stars.‡
- My most technically challenging work was a relatively recent project — an experimental (proof of concept) markdown renderer implemented in terms of Apple's new low-level TextKit2 layout and rendering framework.§

† See my 2019 statistics on this resume or on github.com/colbyn

‡ imager-io/imager

§ SuperSwiftMarkup/SuperSwiftMarkdownPrototype

ABOUT ME

I fell in love with purely functional programming in my late teens and went deep. While not a professional compiler engineer, I've spent years building parsers, modeling data structures, and transforming ASTs — an experience that shaped my approach to solving abstract, technically difficult problems from first principles.

I've taken a nontraditional path — spending most of my twenties pursuing independent technical work. It taught me how to solve hard problems (though not always how to make money from them). Today, I'm especially strong in systems involving Rust and Swift, particularly in domains that demand deep control, like custom TextKit2 rendering or workflows where off-the-shelf tools — including AI — fall short.

HISTORY

2015

Galileo Processing Data Center Tech

Began working there when I was 17, reported to Robert Raver, and then another manager whom gave me a few pay raises for good performance while transitioning the office to the new mainframe build-out.

Basic Networking | Datacenter Misc. | Communication

2016

Uplynk/VDMS (Verizon Digital Media Services) Jr. Developer

Jr. developer in the QA team lead by Asiel Brumfield. I'm not entirely sure when I began

working. I left to pursue a startup with my uncle in encrypted images.

Python | Q/A | video streaming

2018

Secret Startup Experiment

Video | DRM

SubSys/Compiler

GitHub:
`https://github.com/SubSys/Compiler`

A cross-language compiler pipeline that translated **Elm** to **Rust**, implemented in **Haskell** with full constraint based type inference and checking.

Key Features

- Elm → Rust transpilation with semantic fidelity.
- Typed IR modeled and verified in Haskell.
- Explores correctness as a first-class design constraint.

Compiler Engineering | Elm | Rust | Haskell | Type Systems | Experimental

colbyn/commands

GitHub:
`https://github.com/colbyn/commands`

My least loved project but IMO a greatly improved Bash dialect via an indentation sensitive parser. The syntax mimics the clarity of Python, but compiles to raw shell for automation heavy workflows like **AWS deployments**.

Key Features

- Indentation-sensitive syntax for nested blocks.
- Function embedding, aliasing, and multiline strings.
- Supports structured parsing, silent execution, and CLI grouping.

CLI Tools | Automation | DSL | Bash-Inspired

2019

Imager

`imager.io` | `github.com/imager-io`

One of the best image optimization tools on the market.

A modular, open-source image optimization platform designed to outperform commercial tools — with no SaaS dependencies wrapped in a clean CLI and consumable through **Node.js bindings**.

The toolchain included native bindings (`webp-dev-rs`, `ffmpeg-dev-rs`, `x264-dev`, `vmaf-sys`) and a Rust-to-JS bridge via `imager-io-js`. Benchmarks showed over **90% file size reduction** compared to popular SaaS optimizers — without perceptual quality loss.

Image Processing | Machine Learning | Rust | Node.js | WebP | FFmpeg | x264 | VMAF | Performance Optimization

colbyn/web-images-js

GitHub:
`https://github.com/colbyn/web-images-js`

A zero-dependency image pipeline for Node.js — built entirely in Rust and embedded directly in JS workflows. Offers full control over memory, binary size, and output quality without external tools or native runtime bindings.

Key Features

- High-performance image loading and transformation in native Rust.
- Statically linked builds for reproducibility and safety.

Rust | Node.js | Web Development | Performance Optimization | Image Processing | Dependency-Free

colbyn/subscript-old

GitHub:
`https://github.com/colbyn/subscript-old`

I renamed this project to `subscript-old` after I decide to call my note taking tools 'subscript' which seemed more fitting because it dealt with typesetting. **There's actually some pretty cool ideas in here. Abandoning this project is a major regret of my life.**

A data-driven, Rust-native frontend library designed for backend developers — Subscript rethinks web UI as infrastructure. Built for those who want to author views, state, and styles entirely in Rust, with zero XML, full CSSOM control, and expressive compile-time macros.

Ideal for backend developers treating the frontend as just another client — including full inline CSS support (media queries, keyframes, pseudo elements) via a "selector-less functionalized CSS" model. Built-in routing, component messaging, and versioned view syntax enable deeply structured applications without frontend boilerplate.

Feature Highlights

- Inline Rust macros for **media queries**, **keyframes**, and **pseudo classes**.
- **Pattern-matching-like URL parsing** using `parse_url!`, with totality checks and type-safe bindings.
- **Component messaging and subscriptions**, including typed broadcasting and routing by component type.
- Versioned view macros like `v1!` for ergonomic and incremental UI design.
- Zero-runtime-diffing model closer to Incremental DOM than virtual DOMs.

Example Syntax

```
parse_url! {
  [] => {
    Page::Homepage
  },
  ["account", user_id: Uuid] => {
    Page::AccountUser {
      id: user_id
    }
  },
  _ => Page::NotFound
};
```

```
v1! {
  display: "flex";
  button !{
    event.click[] => {
      move || Msg::Increment
    };
    "Increment";
  }
}
```

Rust · Frontend · WebAssembly · Macro Systems · Routing · UI Framework · CSSOM

Impact: Inventive take on frontend architecture through Rust. Encourages architectural unification between client and server codebases, with precise control over styling and rendering.

Milestone

At the end of 2019 I raked up an impressive 1,323 contributions on GitHub that placed me among the platform's most active developers.

To put this in perspective, data from a 2016–2017 GitHub ranking shows the 256th most active user recorded 1,322 contributions—my activity surpassed this benchmark, placing me among the platform's top contributors. While my work wasn't featured in such lists due to followe-

r-based filtering, my focus remains on creating impactful projects.

Sources

Top 2017 GitHub Contributors:
https://web.archive.org/web/20200415010317/https://gist.github.com/paulmillr/2657075/

Colbyn's GitHub Profile (stats at the bottom):
https://github.com/colbyn?tab=overview&from=2019-12-01&to=2019-12-31

2020

Began College

Notable Comments

> *You are such a profound writer and thinker. It has been my privilege to be your instructor of record. One day I will say, I had him in my English class. I have such high hopes for you! Go conquer your world! You're awesome.*
> *—Dr. Jim Birrell*

LaTeX

UVU CS Grader - Computer Science

In my first semester at UVU I took CS1400 by professor Bianca Ruiz, who then offered me a grading position at the end of the semester. I enjoyed this job and wish I stayed (I took trigonometry and calculus concurrently and was expecting subsequent semesters to be just as difficult).

The instructor said my code looked “beautiful” before I signed up as a grader.

Python | Grading | Teaching | Communication

The Subscript HTML Toolchain

GitHub:
`https://github.com/subscript-publishing/subscript-html`

A custom publishing system that reimagines LaTeX for the web.

Academic Publishing | LaTeX-Inspired | HTML | JavaScript | Responsive Design | Note-Taking

My Beautiful Math Notes

school-notes-spring-2020:
`https://colbyn.github.io/school-notes-spring-2020/`

A real-world testbed for Subscript’s HTML toolchain — compiling a semester of math coursework into an interactive, web-native format. Features richly formatted equations, diagrams, and dynamic TOC navigation across topics.

Key Features

- Clean math typesetting with inline and block notation.
- Linked headings and deep navigation for modular study.

Final Culmination (pretty slick):
`https://colbyn.github.io/all-school-notes`
Although looking back I prefer the old style:
`https://colbyn.github.io/school-notes-fall-2022`

Mathematics | Academic Publishing | Subscript | HTML | JavaScript | Interactive Content

Subscript (iPad Edition + Authoring Tools)

GitHub:
`https://github.com/subscript-publishing/subscript`

Content publishing VIA Web-Technologies!

Supports freeform and typed markup content in one medium.

Key Features

- Seamlessly intermix markup with hand drawn content VIA the Subscript Freeform Tools (iPad only).

- Redesigned macOS and iOS editor interfaces from the ground up.
- Native rendering pipeline tuned for complex typeset math, and freeform sketch inputs.

Academic Publishing

LaTeX-Inspired Markup Language | Drawing Tools

JavaScript | HTML | iPad | UI/UX Design

Swift | macOS | iOS

2021

AMI Uploader

GitHub:
`https://github.com/colbyn/ami-uploader`

A lightweight Rust CLI designed to automate the upload of **LinuxKit-generated AMIs** to AWS via S3. It streamlines the gap between image generation and cloud deployment — replacing manual S3 uploads and AMI registration with a single, scriptable command.

Built for reliability in CI pipelines, the tool supports alternate credential injection, customizable AMI naming, and metadata tagging. Designed for infrastructure engineers managing reproducible builds.

Key Features

- One-command upload from local disk to registered AMI.
- Supports name overrides, alternate AWS keys, and region targeting.
- Integrates cleanly into LinuxKit or DevOps build pipelines.

CLI Tools | Rust | AWS | S3 | AMI | LinuxKit

DevOps

2023

punk-lang

GitHub: `https://github.com/colbyn/punk-lang`

A deliberately minimal markup language with LaTeX-inspired syntax and depth-sensitive highlighting — designed to test how little structure is needed to express complex educational content clearly.

Language Design | Educational Content

LaTeX-Inspired | Syntax Highlighting

Markup Language | Experimental

Subscript Freeform Note-Taking App (iOS/macOS)

A handwritten, vector-based note-taking app built from scratch to prioritize **human expression, long-term readability, and semantic structure**. Originally released on the App Store, I later withdrew it to re-architect the data model and rethink some things.

The app uses a custom model space for device-independent rendering, handwritten stroke capture with velocity-aware smoothing (via a Swift port of `perfect-freehand`), and a semantically driven outline system (H1–H6) for navigation and TOC generation.

Subscript reflects a broader philosophy: **separation of content and presentation**, inspired by LaTeX, combined with the authenticity of freeform input—a medium resistant to AI mimicry.

Intro. My Note-Taking App & Why It Matters in the Age of Bots (YouTube): https://youtu.be/PEC5PyNhIds?si=W6w6zOrrK29rD37C

Swift | Native iOS/macOS | Unsafe Rust (FFI)

Vector Graphics | Digital Ink | Document Compiler

2024

Parser & Tree Visualization Toolkit

A cohesive suite of language tooling libraries for **Swift** and **Rust**, focused on parser combinator frameworks and human-friendly visualization of abstract syntax trees (ASTs) and nested structures. Designed to support debugging, inspection, and language toolchain development in functional and systems programming contexts.

Core components include `MonadoParser`[†], a monadic parser combinator framework for Swift; `pretty-tree-rs`[‡], a minimal Rust library for rendering cleanly formatted hierarchical data; and `SwiftPrettyTree`[§], a Swift-native port for readable tree inspection in IDEs or CLI workflows.

Key Features

- Composable, lossless parsers with precise position tracking (`MonadoParser`).
- Compact, dependency-free tree renderers for ASTs and nested structures.
- Interoperable debugging tools for language tooling, REPLs, and test harnesses.

Links:

† github.com/colbyn/MonadoParser

‡ github.com/colbyn/pretty-tree-rs

§ github.com/colbyn/SwiftPrettyTree

Parser Combinators | Swift | Rust | Functional Programming | AST | Debugging Tools | Compiler Engineering | Open Source

3in1Spanish Your go-to Spanish Dictionary, Phrasebook & Flash Cards

A Bilingual Spanish Dictionary, Phrasebook & Flash Card App.

Required a very sophisticated dataset generator that I called the compiler generator. See my YouTube Video for details with commentary.

'How I autogenerate massive (dictionary) datasets with ChatGPT/LLMs and why this matters':
https://youtu.be/nofJLw51xSk?si=WrOwCT7WA6_VTBrO

Prompt Engineering | Data Pipelines | AI Centric Pipelines | Dataset Generation

2025

SuperSwiftMarkdownPrototype

GitHub:
`github.com/SuperSwiftMarkup/SuperSwiftMarkdownPrototype`

While markdown UIs can be trivial to implement the goal here was actually very challenging to do well: provide a rich and intuitive text selection experience (across all GitHub flavored markdown block types) that meets the expectations of iOS and macOS users (an aspect that is very lackluster in the ChatGPT iOS app and especially prior to my work). My proof of concept rendering engine handles text selection, including multi-cursor text selection across and within tables in such a manner that I'd love to one day build a markdown based spreadsheet app upon this technology.

Why did I endeavor to tackle this challenge? (Hint: I didn't do this so that the iOS devs at OpenAI and other companies can steal some of the techniques I developed.) I'm working on a chatbot client based on a branching data model and there are several aspects that while possible is very lackluster as embedded web views. I needed more control that is only possible with native UI toolkits but when it came to advanced text functionality I needed even more control that motivated my early explorations in text rendering.

Swift | Markdown Rendering | iOS | macOS

TextKit2 | Prototype