RhymePad: Seeing the Sound

June 2026

12 min read • 2,594 words

Rhyme is a fact about sound, not spelling. Blue rhymes with shoe and through and you, and not one of them is spelled like the others. Comb, bomb, and tomb all end in the same three letters and not one of them rhymes with another. The ear knows the difference instantly. The eye, staring at the same words, has no idea.

That gap has always bothered me. We write verse with our eyes open and our ears doing all the real work, alone, with no help from the page. So I built a page that listens.

RhymePad is a scratchpad for poets and rappers. You type or paste lyrics into it, and as you write, the rhyme structure lights up underneath you in real time. End rhymes, internal rhymes, slant rhymes, multisyllabic interlocks that thread across word boundaries. It reads the actual sounds, not the letters, and it paints what it hears. I vibe coded the whole thing with Claude Code, talking it into existence one conversation at a time.

Here is the easiest way to show you what that looks like. I asked it to analyze a verse about itself:

RhymePad, on itself
rhymepad.kennethreitz.org

I'm a notepad with a knack for th
I track every syllable and shadow
the cadence in the static, automa
paste a verse in and the rhyme sc

I see the assonance, the consonan
the prominence of dominance in el
internal and infernal, every kern
the perfect and the slant, in a v

color is the family, the brightne
hover any word of yours and watch

Every color is a sound family. Every brightness is how hard the rhyme lands. The whole essay below is really just an explanation of that one picture.

The pad, not the dictionary

There are already good rhyming dictionaries. RhymeZone will tell you everything that rhymes with orange if you ask it about orange. But that's a lookup. You bring it a word, it brings you a list, and you go back to staring at your verse trying to hold the whole sonic structure in your head.

RhymePad inverts that.

> This is the same move as so much of the work I care about: stop making the human translate themselves into the machine's terms. A dictionary makes you query word by word. The pad meets you where you already are, which is in the middle of writing something.

It doesn't tell you what rhymes with a word. It shows you the rhyme architecture of what you're already writing, while you write it. The dictionary is still in there, a lookup panel a keystroke away, but it's the side dish. The moat is the pad. The whole point is that you never have to leave your own verse to see how it's holding together.

Color is which sound, brightness is how hard

The visual language is the soul of the thing, so let me be precise about it.

Every rhyme family gets its own color. All the AY-T words, the tonight / light / flight cluster, share one hue. The next family gets a different one. There's a sixteen-color palette, and colors get assigned least-used-first, so they spread evenly across a verse and two adjacent families never collide into the same shade. You glance at a page and the sound groups separate themselves out by color, the way they already separate themselves out by ear.

Then there's the part I think is the best feature in the whole app: brightness is rhyme strength.

> It's a continuous gradient, not an on/off switch. Two words don't just rhyme or not rhyme. They rhyme to a degree, and the page renders the degree. This is closer to how hearing actually works than any binary classification.

Perfect rhymes blaze. Slant rhymes sit back. Consonance, the faintest kind of echo, barely glows at all. And the gradient runs inside a family too: the perfect anchors of a cluster burn brighter than the loose slant words that have attached themselves to the edges of it.

The effect is that a verse becomes legible at a glance. Bright is where you locked it. Faint is where you're still reaching. You can see the shape of your own craft without reading a single word, the way a producer reads a waveform.

A few quieter signals layer on top. The rhyming word takes a soft tint of its family color, sitting on a translucent block of the same color, so the signal comes through two channels at once. A faint underline marks the exact rhyming tail of a word, the ight under tonight, so you can see not just that two words rhyme but where the rhyme actually lives. Hover over any word and its entire family brightens across the whole page at once, every relative lighting up together, then settles back when you move away.

There are toggles for three lenses: rhyme, alliteration, rhythm. Alliteration underlines shared head-sounds. Rhythm drops a row of dots under the words, sheet music for rapping, where a shared dot color means a shared cadence. You turn on what you need and leave the rest off.

How RhymePad hears a rhyme

RhymePad's whole job is to read what you're writing and color the rhymes as you type. The catch is that rhyme lives in sound, and English spelling is a liar. `orange` and `door hinge` rhyme; `cough` and `dough` don't. So the first thing the engine does is throw the spelling away.

From letters to sound

Every word gets mapped to its phonemes, the actual mouth-sounds, using the CMU Pronouncing Dictionary by way of the `pronouncing` library. `tonight` becomes `T AH N AY T`. When a word isn't in the dictionary, slang, a name, a

word you invented for the bar, the `g2p-en` grapheme-to-phoneme model sounds out a pronunciation, and a pile of lyric-specific repairs patch the rest: `runnin'` becomes `running`, possessives and plurals resolve, and the `-ine` that wants to be `IY N` gets its candidate.

A rhyme is everything from the **last stressed vowel** onward. In `tonight` the stress lands on the final syllable, so the rhyming part is `AY T`, which is exactly what `light`, `flight`, and `write` share. Match on that tail and you've found a perfect rhyme. The quiet trick: RhymePad looks for that match anywhere in the line, not just at the end. That single decision is the whole internal-rhyme engine.

The ladder of looseness

Not every rhyme is perfect, so the engine works down a ladder, each rung a little looser than the last:

- **Perfect**, where the whole tail matches (`creation` / `elation`).
- **Slant**, where just the vowels match and the consonants drift (`hold` / `coal`).
- **Multisyllabic and mosaic**, where the rhyme is a run of vowels that can cross word boundaries. This is where `orange` finds `door hinge`, and where `swimmers`, `finisher`, and `commissioner` turn out to be the same sound stretched over different numbers of words.
- **Consonance**, a shared vowel-plus-consonant ending when nothing fuller is there (`bliss` / `exist`).

Each word carries a strength from whichever rung it landed on, and that strength is what you see on the page: perfect rhymes blaze, slant rhymes sit back, consonance is fainter still. Brightness is the engine telling you how sure it is.

The hard part is knowing what isn't a rhyme

Finding rhymes is easy. Finding rhymes and not finding the ten thousand coincidental near-matches that would turn the page into confetti is the actual work, and most of the engine is restraint.

> Naive vowel-matching produces a page that lights up everywhere and means nothing. The whole game is suppression: knowing what to not call a rhyme. Most of the regression tests exist to defend a single word that was lighting up when it shouldn't.

- `garbage` and `javascript` both carry an `AA` and a schwa, but their endings disagree, a hard `JH` against an open vowel, so they don't rhyme.
- `middle` and `unavoidable` share only a bare schwa-`L` tail, which half the dictionary has. Too weak to count.
- `smell like a` looks like it rhymes `myself why`, until you notice it dangles on the article `a`. The real rhyme was `smell like`, and the trailing word adds nothing.
- A word repeated at line ends is a rhyme. `again` / `again` / `again` is a real monorhyme scheme. But the same word duplicated whole-line, or leaned on as a four-times hook, is a refrain, not a rhyme, and stays dark.

The rule behind all of these is the same: color a rhyme only when a listener would actually hear one.

Context, not just pairs

A rhyme isn't a property of two words in isolation. It's a property of where they sit, so the engine reads the room. Rhymes don't reach across a blank-line stanza break, because each verse is its own world. A vowel family stays local, because a shared sound a hundred lines apart isn't a rhyme, it's a coincidence. Words a singer leans on as a refrain get muted so they stop shouting over everything. And a handful of dialect mergers are folded in the way rappers actually deliver them: the `near` vowel, the cot-caught merger, nasal and plural endings treated as the soft inflections they are.

The result is an engine that knows `orange` rhymes with `door hinge`, that `garbage` and `javascript` don't, and, most importantly, the difference between the two.

The tests are the spec

Almost every real bug lived in the order of those passes. They run perfect first and loosest last, and a greedy early pass will swallow a word that belonged to a tighter rhyme three lines down. The phoneme matching is arithmetic. The ordering is judgment, and judgment is where things break.

There is no formula for "a rhyme a listener actually hears." You can't derive it from first principles, because it isn't a principle, it's a thousand small human verdicts. So I built it the only way that holds: against real verses, one correction at a time. I'd paste in a verse, find the single word lit the wrong color, fix the engine until it was right, and then, before touching anything else, freeze that exact verse into a regression test.

> This is the inversion of normal spec-driven work. There was no spec to implement first. The corpus of real verses became the spec, accreting one judgment at a time. Every test is a frozen sentence that says: these words rhyme, those don't, and here is the proof.

There are around eighty-two of those now, each one a real bar from a real song, validated against people who do this for a living: Lil Wayne, Eminem, MF DOOM, Drake, Big Sean, Kanye, with T.S. Eliot and Tool folded in so the ear behind the engine wasn't tuned for only one tradition. The hardest single test is DOOM's "Doomsday," about the densest interlocking rhyme in recorded rap. It doesn't break the engine. The families separate cleanly and you can watch the architecture DOOM built by ear hold together exactly the way it sounds like it should.

That suite is the only reason the thing works at all. The whole app was vibe coded, every line written by describing what I wanted to Claude Code and steering, never typed by hand. I love working that way, but a rhyme engine is exactly where it should fall apart, because the tuning is the worst kind of fiddly: a fix that catches one rhyme quietly breaks three others ten verses away, and you would never catch it by eye.

> This is the part of vibe coding nobody oversells enough. The model is happy to make a confident change that breaks something subtle three files over, and it will tell you it's fine. The test suite is what converts that confidence into something you can actually trust, because it answers, in a second, the only question that matters: did this quietly wreck something that used to work.

Vibe coding without tests on a problem this subtle would be building on sand. With the suite under it, every change to pass ordering got an instant green-or-red answer. I could let the AI rewrite an entire pass and know within a second whether I'd just regressed Eminem. The tests are what kept the vibe coding honest. They turned a domain that punishes confidence into one where I could move fast without lying to myself.

The keepers earn their place

Here's the part of building it that taught me the most, and it's not a feature. It's everything that isn't one.

I kept building elaborate things and then cutting them back to the simplest version that felt right.

> This is the design rhythm I trust most by now: build the full ornate version, live with it long enough to feel what it costs, then strip it to the one move that earned its keep. You can't always see the keeper until you've built the thing around it that doesn't work.

There was a constellation weave, glowing threads drawn through every member of a rhyme family across the page. I built the whole thing. It was beautiful and it was noise. I stripped it down to the hover-illuminate, which does the same job, find the family, with none of the clutter. Grammar and part-of-speech highlighting got tried three different ways and removed all three times. Beat-arcs, right-margin connectors, dual-family coloring on words that belong to two clusters at once: built, then cut.

Every cut made the app better. The features that survived survived because they earned it, and the ones that didn't were taking attention away from the verse, which is the only thing on the page that matters. With AI you can build the elaborate version fast enough that subtraction becomes the real design work. The keystroke is cheap now. Knowing what to delete is the whole skill.

Where it goes next

Honestly? I'm not sure. I don't have a roadmap and I've stopped pretending I do.

There's one gap I can name. So much of RhymePad lives in the hover, point at a word and its whole family lights up across the page, and a finger on a screen doesn't hover. On a phone you get the colors but not the conversation with them, and that's probably the thing I'd look at first.

After that it goes pleasantly vague. Maybe it reads a verse back in the meter it detected. Maybe the dialect coverage gets deeper. Maybe none of that. But I don't think the next step is really a feature at all. It's the same loop that built the engine in the first place: someone pastes in a verse, finds one word lit the wrong color, and tells me. Each one of those is a new test and a small correction, and that's how the ear behind the engine keeps getting sharper. The roadmap is just the next verse that breaks it.

What it's for

RhymePad doesn't write your verse. I want to be clear about that, because it's the whole philosophy. It won't finish your line or suggest your next bar or generate a hook. The closest it comes is a dotted-gold mark on a dead line ending that sits one phoneme short of a rhyme, and even that only points at a gap you made and trusts you to fill it.

What it does is show you your own work. The rhymes you locked, the ones you're reaching for, the buried interlock you put there by instinct without quite knowing you'd done it. It takes the thing your ear already knew and renders it where your eye can finally see it, so you can edit with both at once. It's of a piece with the poetry platform I built for Sarah, and with everything I keep coming back to: the machine handles the how, the human keeps the what and the why, and the craft stays yours.

The ear always knew the architecture was there. The page just learned how to listen.

RhymePad is live at rhymepad.org. More on the project here.