Ex 2.1.1: un-anchored color-mixing transformer

> A small transformer learns a character-level language of color-mixing equations, solving the forms it saw in training and unseen hex pairs. Color turns out to be linearly decodable from its residual stream, with each seed putting *redness* in a different place. Held-out *named* pairs sit at zero accuracy.

In M2, we want to see whether Sparse Concept Anchoring carries over from autoencoders to transformers. Before we anchor anything we need a baseline, so this experiment trains a small transformer on a well-defined task.

The model must learn a character-level language of color mixing equations on a 16-level RGB grid. Here are the sample types, which we will refer to throughout:

Type	Example
Named pairs	`red + blue = purple`
Hex pairs	`#f00 + #00f = #808`
Cross-form	`red + #00f = #808`
Alias	`red = #f00`

Every operand spans several tokens in both of its spellings, to force the model to perform two tasks simultaneously: it must mix the colors and spell the result.

Mixing (`+`) is the channel-wise round-half-up mean, so each prompt has one correct completion.

We sweep width {16, 32, 64} × depth {2, 4} × 3 seeds (experiment definition), and for each condition we measure:

- Completion accuracy: greedy decoding, scored as an exact string match, over four evaluation sets. Those are named pairs seen in training; held-out named pairs, which never appear as named equations, so the model has to combine the alias dictionary with hex arithmetic to answer them; hex-only equations; and cross-form operand pairs that were never shown together.

- Probe alignment [1]: ridge regression from the residual stream at each depth out to the operand color, the result color, and the *redness* of the result.

Hypotheses

H1. A small nGPT should learn the task, with near-perfect accuracy on seen forms and on unseen *hex* pairs; that leaves the anchored runs room to show any degradation later.

H2. Color should be linearly readable from the residual stream, more so as depth increases.

H3. The *redness* probe directions should vary from seed to seed. This is part of the motivation for this work: searching for a concept after training turns up a different geometry every time, whereas SCA should let us fix the location in advance.

Training data

The corpus sampler is deterministic. Regenerating it here with the constants defined in the experiment gives back the same training data the model saw, and these are its first lines:

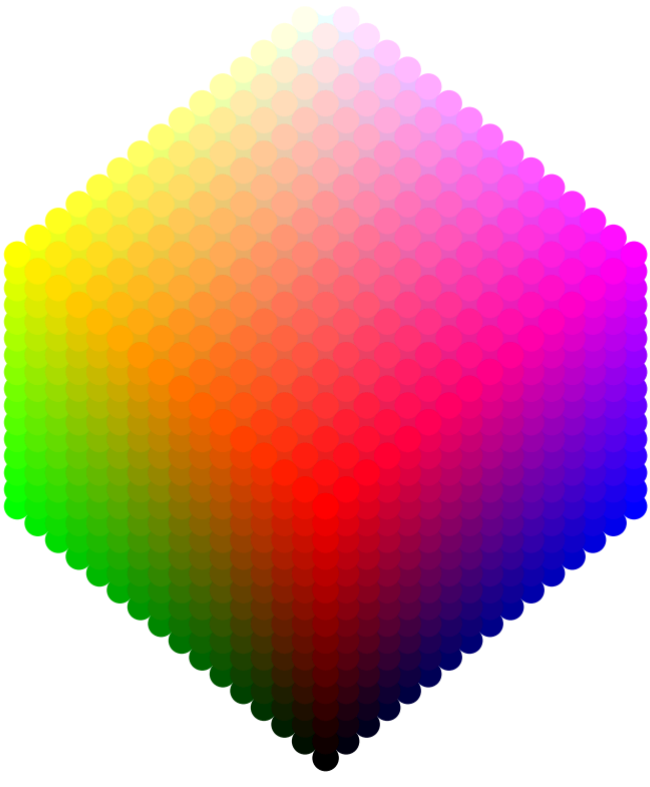
```
chartreuse + violet = gray
    #076 + #c0b = #649
    #1bc + #196 = #1a9
    #0c1 + #33a = #286
  lavender + #fc8 = #cac
       mint = #8f8
   white + white = white
   yellow + cyan = mint
    #4f7 + #be3 = #8f5
     lavender = #88f
```

40,000 lines in total: 23,924 hex, 6,078 named, 6,101 cross, 3,897 alias.

Between them they cover 29,875 distinct operand pairs, **0.36%** of the 8.4M in the grid. So the unseen-pair eval sets, sampled to steer clear of all of them, test the mixing rule rather than recall.

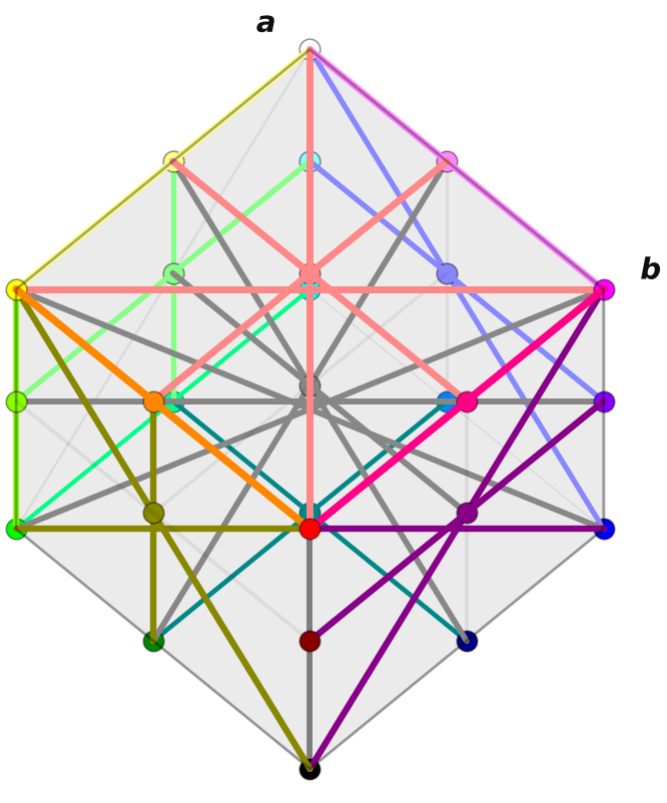
The color space

Every color is a point on an RGB grid with 16 levels per channel, so $16^3 = 4096$ points in all. If we rotate the cube so its black-to-white diagonal stands vertical, *value* runs up the page and hue wraps around it. That is the figure below, seen front-on toward the *red* corner. Hex and cross equations draw their operands from anywhere in this cube.

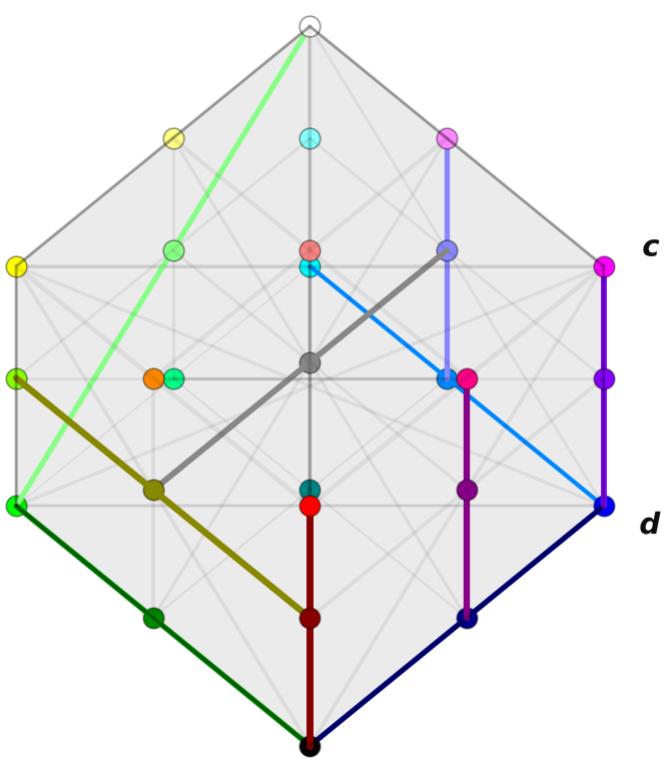


The 16^3 hex grid

The 27 named colors sit only on the $\{0, 8, 15\}^3$ sub-lattice, the corners and edge midpoints of the cube. Note that no *color* is held out: every point shows up in training, since hex operands are sampled over the whole grid and each name appears in an alias line. What we hold out is operand pairs, both named and hex.



Train



Held out for eval

*Named pairs on the cube. **a-b**: white + magenta = orchid. **c-d**: magenta + blue = violet.*

Both figures show the same lattice from the front. The vertices are the named colors, and each edge joins the two operands of a named pair. The midpoint of an edge is the answer that equation should produce.

Two edges are labeled as worked examples:

- $\overline{ab}$: white + magenta = orchid (train)
- $\overline{cd}$: magenta + blue = violet (held out)

Only the connected pairs ever appear as named equations *with a named answer*. Every other operand pair the model sees is written in hex or cross form, and those draw their operands from the full 16^3 grid.

A held-out edge like `magenta + blue = violet` can be answered two ways. One is recall, which is ruled out here, since that named rendering never appears in training. The other is composition: recognize both names as colors, mix them as if they had been written in hex, and translate the result back into a name. Composition is what the `named_holdout` eval set measures.

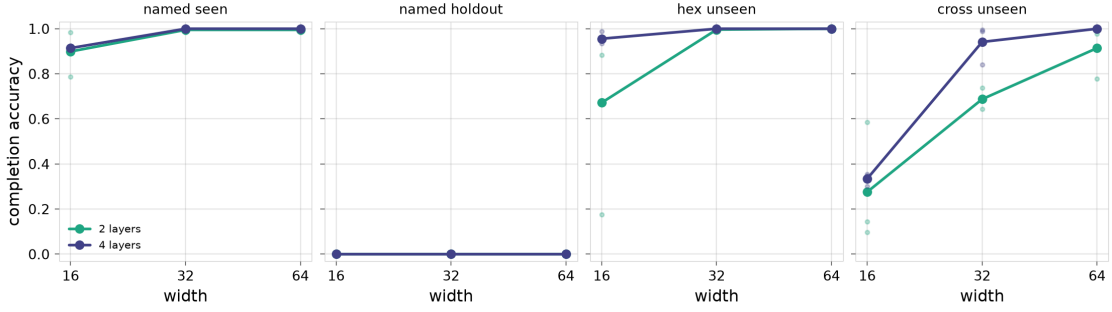
The `hex_unseen` and `cross_unseen` sets are sampled at evaluation time from the full grid, steering clear of every operand pair the corpus used.

Results: Accuracy on unseen hex pairs spans **0.18–1.00** across the sweep, while held-out named pairs, the compositional test, span **0.00–0.00**. The figures below break this down by condition and eval set.

Completion accuracy across the sweep

The figure below shows accuracy vs. model width, with one panel per eval set. Each panel has one line per model depth (the mean over seeds). Individual seeds are shown as faint points.

The named-holdout panel is interesting. It can only be solved by combining the alias dictionary with the mixing arithmetic, and we find that the model never learns to do that.

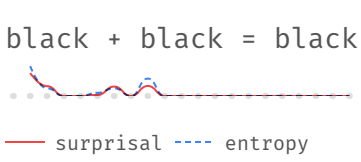


Watching it answer, character by character

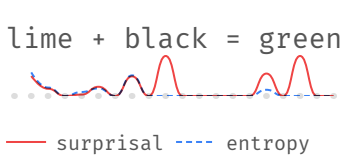
Let's see where in each sequence the model was unsure.

For the d64-L4 model (seed 0), we plot one example per eval set and draw two series beneath the text, both as fractions of $\log |V|$, the value a uniform guess over the vocabulary would give. The first is the surprisal of the model at each character: how "surprised" it is by the character that actually comes next. The second is the entropy of its predictive distribution, the surprisal it expected on average before seeing that character.

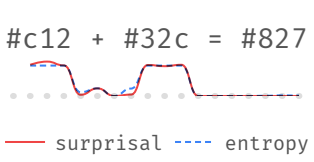
Operands are unpredictable, so both series should spike at the first characters of each operand and settle as the prefix pins down the rest. Everything after `=` can be computed from the operands, so a model that has learnt color mixing should coast through the answer at near-zero surprisal, even on operand pairs it has never seen. If instead it guesses the answer, surprisal should spike across the answer characters.



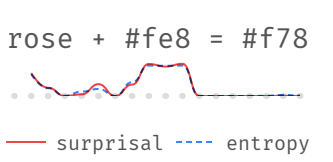
named_seen



named_holdout



hex_unseen

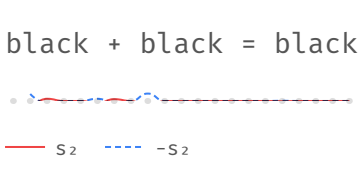


cross_unseen

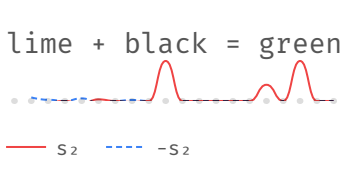
The gap between those two series is the surprisal beyond what the model expected,

$$s_2 = \frac{i - h}{\log |V|}$$

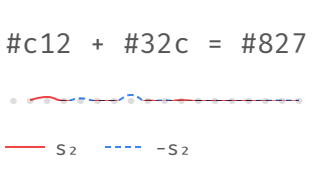
where i is the surprisal and h the entropy. This sits near zero when the model confidence matched the outcome, whether it was confident and right or unsure and appropriately surprised. It goes positive when the model was confidently wrong, and negative when the character was more predictable than its distribution suggested. The sparkline clips at zero, so we draw the negative values as a second, flipped series, $-s_2$.



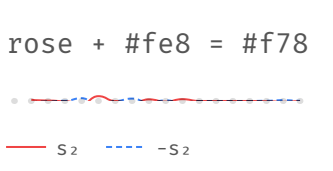
named_seen



named_holdout



hex_unseen



cross_unseen

None spike except `named_holdout`, the one set this sweep never solves (accuracy 0 above). The model is confident even on those wrong answers: entropy stays low while the true characters arrive as a surprise. What is the model so sure of?

Why the named answers fail

The sparklines above are teacher-forced: the model is shown the true answer from the validation set, and we watch how much each one surprises it. `lime + black = green` is an interesting case.

Left to choose, this seed opens `lime + black` with `t`, for *teal*, so the true `g` is mildly surprising. Once it is forced onto `g`, the model guesses `r` correctly, since *gray* and *green* share the prefix `gr`. Then the true `e` is very surprising: on the `gr…` branch the model is all but sure the word is *gray*, but `e` rules that out. The spike is the model fluently spelling a different palette name, then being surprised when the truth arrives.

Teal is a one-channel neighbor of the true mix, and the tall spike on the `a` of `black` hints at why. After `lime + bl` the model puts 100.0% on `u`, so it is all but certain the second operand is *blue*, and `lime + blue = teal` is an equation it trained on.

When it sees that the operand is different, the correction barely reaches the answer. Going from the `lime + blue =` prompt to `lime + black =` lifts *gray* by a factor of 74, from 0.2% to 13%. The true answer *green* moves by a similar factor and stays negligible, from 2e-08 to 2e-06, while the trained *teal* keeps the top spot at 84%. So the operand correction redistributes mass among wrong names rather than finding the arithmetic. The model does learn the result-form rule, which says a named answer appears iff both operands are named. The difficulty is choosing which name, and a trained neighbor seems to overrule.

Below is every held-out pair, prompted exactly as in the `named_holdout` eval set, with one column per seed of the chosen architecture.

prompt	expected	seed 0	seed 1	seed 2
`blue + black =`	navy	black	black	black
`lime + black =`	green	teal	gray	teal
`black + red =`	maroon	purple	olive	purple
`rose + navy =`	purple	gray	gray	gray
`blue + cyan =`	azure	teal	black	black
`blue + magenta =`	violet	lavender	gray	lavender
`orchid + azure =`	lavender	gray	gray	gray
`white + lime =`	mint	gray	chartreuse	lime
`chartreuse + maroon =`	olive	gray	gray	maroon
`olive + lavender =`	gray	lavender	lavender	lavender

Greedy completions of the `named_holdout` prompts, d64-L4, all seeds.

The model never answers these in hex. It always reaches for a name, and usually one adjacent to the true mix: 25 of 30 guesses land in the one-step neighborhood of the mix, against 17% for a name drawn uniformly from the palette. Sometimes the name is an echo of one operand (`olive + lavender = lavender`), though on this palette an operand is often a neighbor anyway, so that part is weaker evidence than it looks.

The seeds mostly agree: 4 of the 10 pairs draw the same wrong answer from all three. Independent guessing inside the neighborhood would produce well under one such pair, so the bias is systematic; perhaps retrieval of the nearest memorized named equation.

The mixing arithmetic itself is fine. Prompted with the same held-out value pairs, seed 0 solves **10/10** in hex form and **10/10** in cross form, against **0/10** as named equations.

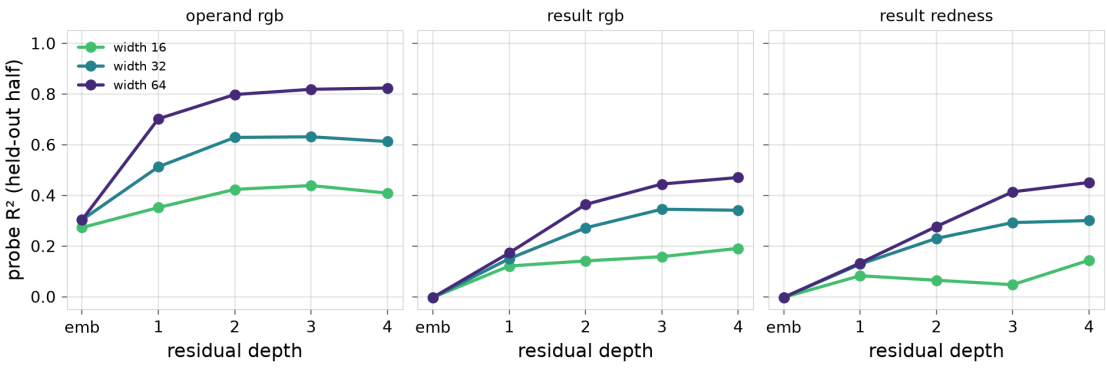
The corpus may make this hard for the model. Possible causes:

- Named equations draw on only 66 distinct pairs, so each one is seen about 91 times in training. A lookup table is enough, and the model may build one (`named_seen` ≈ 1). Once the loss on that slice reaches zero, nothing nudges the model toward the compositional route.
- The alias dictionary runs one way. Alias lines are always `name = hex`. This may be an instance of the *reversal curse*, where training on `A = B` doesn't teach `B = A`.
- A hex answer can be computed channel-by-channel, whereas the first character of a name depends on all three channels and the inverted dictionary at the same time. The probe section below looks at this more.

A few corpus changes might help: reverse some alias lines (`#f00 = red`); add named operands whose off-palette mix forces a hex answer (`red + navy = #804`), so that `name + name` prompts have to engage the arithmetic instead of the lookup table; and use a denser named palette, so memorization is harder.

Where color is represented

Here we fit the probes at each residual-stream depth (depth 0 is the embedding) and plot their R^2 against depth[2]. The figure has one panel per probe target and one line per width. We test only the deepest models (L4) and show the mean over seeds.



Rising R^2 for the *result* means the mix becomes partly readable before the answer starts. It plateaus well below the operand R^2, though, even in conditions whose hex accuracy is perfect. Probing every answer position, per channel, would map that spread-out schedule (to do).

The probes read the residual stream at two positions, marked below. The first is the last character of the first operand, where the whole operand has been read in, so its value can be represented. The second is the space after `=`, the last position before the answer begins, where the result has to be ready.

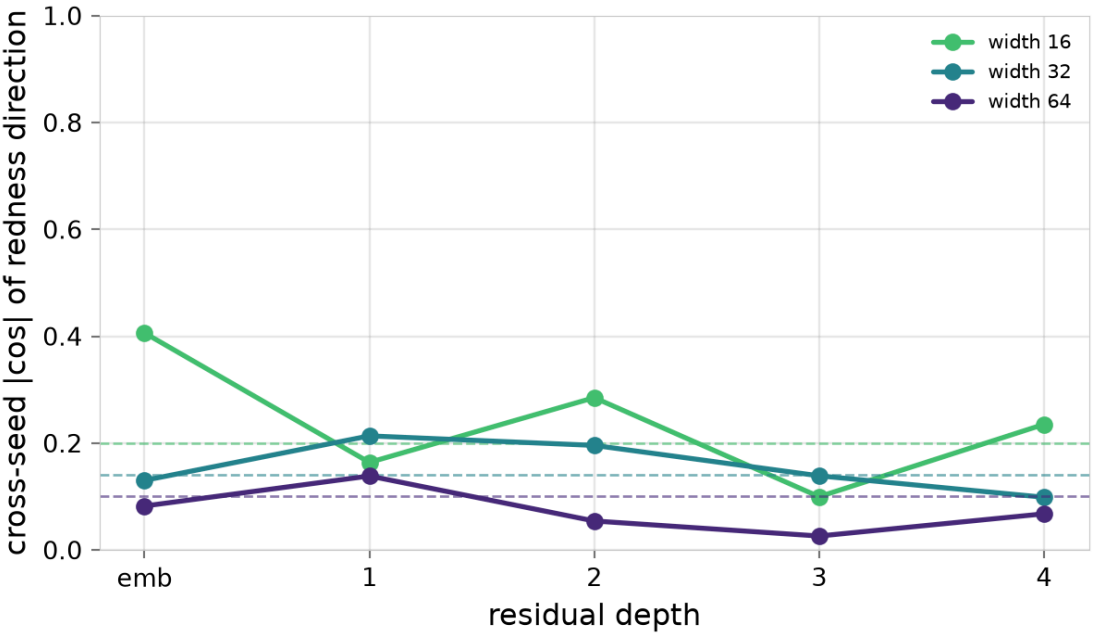
```
red + blue = purple
orange + #2c7 = #9a4
```

operand probes read the color of the first operand at this character; result probes read the result color and redness at the space just before the answer (shown as ␣). The dimmed answer is never probed.

Do seeds agree on where *redness* points?

For each pair of seeds trained with the same architecture, we take their fitted redness-probe directions and measure the absolute cosine similarity between them, layer by layer. Two random directions in n dimensions should have $|\cos| \approx 0.8/\sqrt{n}$, drawn here as the dashed line.

If this geometry were stable across seeds, there would be little point in anchoring. The spread we see is part of why we want to pin the direction down at training time.



Findings

The smallest condition that saturates the unseen-pair eval sets is **width 64, 4 layers**. For D2.1, we can take that architecture as the baseline and add the anchor, which pulls sequences labeled *red-ish* (supplied as sparse, noisy labels) toward a chosen direction at chosen layers. Then we can re-run these measurements and compare the anchored and baseline versions.

The held-out named pairs sit at zero validation accuracy, so that set gives the anchored runs no headroom and probably can't help us spot any unintended degradation.

---

1. A probe is a small linear model we fit on the internal activations of the model to read out what those activations carry. Ridge regression is linear regression with a penalty on large weights, which keeps the fit stable. ↩

2. This R^2 is the fraction of target variance the probe recovers, so 1 means the color is fully readable from the stream and 0 means it is not there linearly. ↩