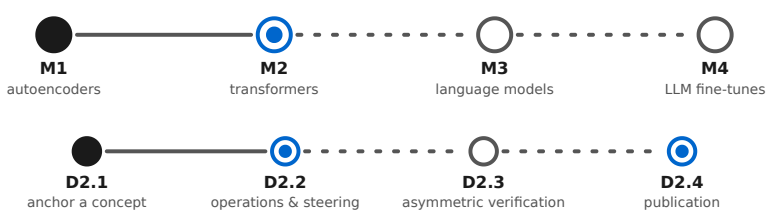
Sparse Concept Anchoring in transformers

Sparse Concept Anchoring (SCA) is a training-time method for concept control. A light geometric regularizer, driven by a small number of noisy labels, guides a chosen concept toward a known location in representation space. The geometry is shaped during training, so there is no need to reverse-engineer it afterwards: the concept lives where you put it, and the side effects of suppressing or ablating it can be bounded before running the intervention.

The first milestone (M1) established the method in autoencoders: paper, blog post, code. This site holds the experiment reports for the second milestone (M2), which asks: does SCA transfer to transformers? We train a small transformer to do vector math and anchor concepts in its residual stream. The vectors are colors, in a synthetic color-mixing task (`red + blue = purple`) where ground truth is unambiguous, so a negative result stays interpretable. The plan and deliverables are in the project README.



- Experiments
- Iteration 0 (prep)
- D2.1: anchoring in a transformer
- D2.2: anchoring an operation
  - Removing *red*
  - A richer grammar for *red*
  - Anchoring an op word, and the pivot
  - Teaching the in-context task
  - Anchoring and removing the inferred op
  - What the edit does, and how we measure it
  - A recipe that keeps the anchor on every run
  - The recipe of record, scored for the post

Experiments

Each report is a literate script.[1] Each entry has searchable tags and a strip of the report's figures, in reading order.

1 A literate script is plain Python with Markdown prose between the cells.…

Iteration 0 (prep)

These experiments were preparation for the main work: exercising the infrastructure, testing the normalized transformer architecture, and tying off some loose ends from M1.

▼ **Reports**

nGPT scaling

Width × depth sweep of our simplified nGPT to check that it stays well-behaved as it grows. Converged loss improves with width and is flat across depth, and no condition spikes or stalls. The architecture seems safe to build the color-mixing experiments on.

`ngpt` `scaling` `architecture` `infra`

M1 2.9.1. Deleting *red*, now in JAX

We ported the main M1 result from PyTorch to JAX: Anchor *red* to one latent axis of a small autoencoder, then zero the axis and watch red disappear. The result reproduces, seed sensitivity and all, and our new infrastructure holds up.

`anchoring` `ablation` `jax-port` `seed-sensitivity`

M1 2.9.2. Fallback control

We compared two remedies for ablation seed-sensitivity: **1.** optimal ablation, which picks a replacement constant after training, and **2.** fallback control, a decoder-only loss term that teaches the model what to output once the concept has been removed. Fallback control worked well. Neither remedy touches the other half of the variance, where anchoring itself fails.

`ablation` `fallback-control` `optimal-ablation` `seed-sensitivity`

M1 2.9.3. When anchoring fails

Every failing run anchors first and then comes apart during the high learning-rate plateau. Halving the peak learning rate removes the failures.

`anchoring-failures` `training-dynamics` `learning-rate schedules`

M1 2.9.4. Closed-loop regularizer weights

We replaced the timed regularizer anneal with a feedback controller, so that each weight climbs while its constraint is being violated and settles back once it is met. It kind of worked but not very well, and it adds complexity.

`feedback-controller` `regularizer-weights` `schedules`

D2.1: anchoring in a transformer

D2.1 asked whether SCA works in a transformer at all. We designed a task where a small transformer learns the geometry of color on its own, then anchored *red* to a chosen direction in its residual stream. With a repulsive term, a schedule for it, and a pull that finds the red operand by itself, *red* landed where we put it, graded by how red each color is, at no measurable cost to the task.

▼ **Reports**

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.

`char-tokens` `hex-colors` `color-names` `linear-probes` `generalization`

2.1.2. Making composition necessary

The previous model never answered a held-out *named* pair, so we changed the grammar to make composition the only route: reverse alias lines, and named equations whose mix falls off the palette. The model picks up both new skills and still won't chain them within one forward pass, leaving held-out named accuracy at zero.

`char-tokens` `hex-colors` `color-names` `composition` `grammar-design`

2.1.3. Named colors only

No hex codes, just one opaque token per color and nothing in the text to say that colors are values at all. The model infers the geometry from co-occurrence alone: its embeddings hold the RGB cube as a linear subspace, it computes mixes in value space just before answering, and its held-out guesses land on or beside the right color.

`word-tokens` `named-colors` `embedding-geometry` `vocab-size`

2.1.4. Spelling the names

Does geometry inference need one token per concept? Not at 216 colors. We test the same equations but with every color spelled as an opaque four-letter name. There's a small drop in held-out accuracy, but the value subspace survives. Reading names takes most of the network's depth, leaving one layer for the mix to live in. The 27-color grid is too coarse.

`char-tokens` `spelled-names` `embedding-geometry` `depth`

2.1.5. Disjoint vocabularies and more named colors

Names and hex codes. Both sublanguages train, and each builds its own linear color geometry. But the two geometries resist being combined, even under pressure from a narrow residual stream. Keeping them apart apparently costs less than merging them would.

`char-tokens` `hex-colors` `long-color-names` `disjoint-vocabularies` `embedding-geometry`

2.1.6. Anchoring *red* in a transformer

Our first anchored transformer, with a single attractive term. The anchor moved the activations to the chosen direction, and cost the task nothing measurable. But it moved the whole color cube, rather than *red* in particular.

`word-tokens` `anchoring` `selectivity`

2.1.7. A repulsive term and a narrower pull

We tested two mechanisms to improve anchor selectivity: **1.** Apply the anchor term only to operand 1 (no other tokens), and **2.** Add a repulsive term to clear the target subspace. Both work, but 1. worked better, and their effects stack somewhat.

`word-tokens` `anchoring` `repulsion` `selectivity`

2.1.8. Repulsion tuning

We tested anti-subspace schedules (trailing timing and strength) to find an operating point that contains the cube-wide drift without reducing selectivity. Holding the repulsion high for longer contains the drift and keeps the margin, which nothing before this did. It's unclear if the response is well-graded.

`word-tokens` `anchoring` `repulsion` `schedules` `grading`

2.1.9. Softmin sequence pooling

Anchoring on the first operand alone worked best so far, but in natural language we won't know which tokens hold the concept. So we let the pull choose its own position, by pooling over the line with a soft minimum. It finds the first operand unaided and the anchor stays healthy, though only the softest pooling stays graded in every run.

`word-tokens` `anchoring` `pooling` `mellowmax` `grading`

2.1.10. A label that doesn't point

In ex-2.1.9 the label still only marked the first operand, so the label itself said where the concept was. Here a line is labelled when either operand is red, and the pull has to find the red one line by line. It does, and the anchor is about as selective as when we told it the position.

`word-tokens` `anchoring` `pooling` `weak-labels` `selectivity`

2.1.11. Ablations and a survey for the D2.1 operating point

A survey. The schedules and weights in the recipe were inherited piece by piece and never tuned together, so we tried removing each one, then searched over what was left. The anchor schedule and half the training could go, but the schedule for the repulsive term could not. The search found a wide region of good settings and proposed a stronger anchor at its edge.

`word-tokens` `anchoring` `ablation-study` `sobol-search` `schedules`

2.1.12. Fitted channel probes over the anchored checkpoints

We fitted linear probes to read each operand's color at every depth and position of the D2.1 models. Anchoring costs no color readability that we can measure. The probes also explain an odd pattern in the D2.1 figures, where positions before the second operand seemed to respond to its color: it comes from which test equations we used, not from the model.

`word-tokens` `linear-probes` `checkpoints` `decodability`

D2.1 figures for the anchoring post

Figures for the LessWrong post on D2.1, drawn from the published results above: the anchoring recipe built up one piece at a time, and where in the model the response sits.

`figures` `grading-clouds` `blog-post`

D2.2: anchoring an operation

D2.2 anchors an operation rather than a color. Since the pivot, the model infers which op a line uses from a few solved examples, so the op has no word of its own, like the abstract concepts we care about in language models. The anchor lands at little cost to the task, and removing the anchor direction takes the op out gradually. Under the edit, the model moves its answers onto the other ops about as much as an ideal predictor would, though context by context it doesn't always pick the same ones. On the recipe of record, which pools the pull over blocks 1 to 3, the anchor held on every run at fresh seeds, the edit removes most of the op, and the other ops stay within the selectivity criterion on every seed. A model trained without the op lands only a little closer to the target null[†] than the edited model, so most of the distance left is what a model of this size does on contexts like these. The design doc has the recipe, the claims, and what comes next.

Target null. Where a run *should* land.

Removing *red*

D2.1 anchored *red* but never removed it, so D2.2 began by removing it. Projecting out the anchor direction took *red* out, and took out more where the line was redder, at a small cost to lines with no red. Of the edits we tried, the plain projection stayed the best choice.

Reports

2.2.1. Suppressing *red* in the anchored transformer

Our first intervention on an anchored transformer: remove the anchor direction from the D2.1 models. *Red* goes, more so the redder the line, and the damage stays within the bound the geometry sets in advance. A little of the cost lands on lines with no red, through the `+` and `=` tokens.

`word-tokens intervention suppression checkpoints eval-contract`

2.2.2. A designed response to suppressing *red*

We tried the fallback training from M1 on the transformer: a term that teaches the model to answer as though *red* were *mid-gray* once *red* is removed. Every seed gives that answer under the edit it was trained with, though it carries over only partly to the plain projection. That edit also hurt lines with no red, in every anchored model, with or without the new term.

`word-tokens intervention fallback training eval-contract`

2.2.8. A survey of the intervention operator on the stored ex-2.2.3 checkpoints

A survey of edits on stored models, with no training. Projecting out the anchor direction everywhere was already selective enough. A threshold that spares weakly aligned states removed less *red*, and was no more selective that we could measure.

`survey intervention eval-contract`

How far the answer moves: an RGB-distance readout beside expected exact match[*]

Expected exact match. The score a run *at the chance that is drawn or not* against with a row seen.

Exact match gives no partial credit, so we re-scored stored models by how far the answer moves on the color grid. Answers that lose *red* move between half and three quarters of the way to a random guess, and lines with no red barely move. The distance also varies much less between seeds.

`reanalysis probes methodology intervention`

A richer grammar for *red*

Before we could anchor an op, the grammar needed more than one, so it grew from one op to eleven. We anchored *red* again at each step, and the recipe held. Removal was clean on every op but one, where the model kept about a quarter of the answers that needed red.

Reports

2.2.3. The multi-op grammar, with *red* anchored again

The grammar grows to six ops (`mix`, `add`, `screen`, `multiply`, `lighten`, and `darken`), and we check the D2.1 recipe on it. The stronger anchor from the ex-2.1.11 survey put the anchor direction on the `=` and op-word tokens, so removing it broke lines with no red, and we stayed with the D2.1 recipe.

`word-tokens multi-op regression survey-handoff`

2.2.4. A scouting round before the anchored-op experiments

A scouting pass over candidate ops, with no training. Ops that act on hue, saturation, or value spread their answers through the color cube, and they are the first where the order of the operands matters. We proposed adding six new ops and dropping `add`.

`scouting multi-op grammar`

2.2.5. A pilot of stochastic rounding

A pilot of rounding answers at random, in proportion to where they fall between grid levels. The model learns the coin flip itself, so no model can match every rounded answer, and we score the probability it puts on the right answer instead. Anchoring is unaffected.

`pilot multi-op grammar eval-contract`

2.2.6. A pilot of the whole-span labeller

A pilot of a label that pulls the whole line rather than just the operands. Some of the pull moves onto the answer at no cost, but removing the anchor still leaves those answers as they were, because the model has already decided the answer one position earlier.

`pilot multi-op anchoring`

2.2.7 A pilot of the syntax embeddings

Removal costs something on lines with no red because the `=` and op-word tokens pick up part of the anchor direction. This pilot asks why. The output side of the shared token table puts it there. Giving the model a separate output table moves it off those tokens at no cost to the task.

`pilot multi-op anchoring selectivity`

2.2.9. The grammar handover

The changes from the scouting round and the pilots, together: eleven ops, random rounding, the whole-line label, and a separate output table. *Red* still lands and the task is unhurt. Removal was clean on eight of the eleven ops but fell short on three that take only the hue, saturation, or value of one operand, so we didn't adopt the combined setup as it stood.

`prereg multi-op anchoring selectivity`

2.2.10. Three reads before the handover re-run

The removal misses came from how we chose the lines to score: removal acts on *red* like a change of hue, and the lines that missed needed only its saturation or brightness. We proposed choosing the scored lines by hue instead.

`scouting multi-op intervention anchoring`

2.2.11. The handover re-run

Ex-2.2.9 again at fresh seeds, with the scored lines chosen by hue. *Red* lands, stays through training, and comes out cleanly on ten of the eleven ops. On `hue-hsv` the model keeps about a quarter of the answers that needed red, so we still didn't adopt the setup as it stood.

`prereg multi-op anchoring selectivity`

2.2.12. What the stream holds on `hue-hsv`, and a small recipe sweep

On `hue-hsv`, the answers that survive removal come from reds that lean neither toward orange nor toward pink, and the model rebuilds them inside its blocks after the edit. None of the changes to the recipe in a small sweep removed more.

`scouting multi-op anchoring selectivity`

2.2.13. Does a heavier anchor make the leftover predictable?

Does a heavier anchor make the leftover on `hue-hsv` the same size every time, so we could measure it once and subtract it? It doesn't: across a wide range, the anchor weight hardly changes anything. Anchoring *red* to a plane instead of a single direction made the leftover a little smaller, but not by enough to justify the extra room, so the leftover stays a known side effect.

`anchoring selectivity multi-op methodology`

Geometry under anchoring: a whole-geometry read over the stored runs

A re-analysis of stored models from three experiments, comparing how anchored and un-anchored models arrange their colors. Past the first block, anchored models arrange them differently from the controls, and more alike from seed to seed. A light anchor mostly adds a direction, while a heavier or longer one moves the deeper geometry away from the RGB cube.

`scouting anchoring geometry representations`

Where the op1 lean sits: a reanalysis of ex-2.2.11's stored runs

A re-analysis of why the first operand leans a little toward the anchor direction. With its own output table, the model uses that direction to predict that an op word or `=` comes next, and that accounts for more than half of the lean. The whole-line label adds a little at most.

`scouting anchoring containment readout`

Anchoring an op word, and the pivot

Ex-2.2.14 anchored the op `difference`, and it landed, but almost entirely on the *word* `difference`, so removing it would be hard to tell apart from deleting the word. The concepts we care about in language models mostly have no word of their own, so we took the op words out: each line now shows a few solved examples and a query, and the model has to infer the op.

Reports

2.2.14. Anchoring an operation

Our first anchored operation: `difference` in place of *red*. It lands more firmly than *red* did and costs the task nothing, and every other op anchors the same way. But nearly all of it sits on the word `difference` itself, and little reaches the positions where the op is used.

`anchoring operation smoke-test preregistration`

2.2.15. Lines cut short by the training window

Training windows cut lines at their edges, and the anchor asked the visible part of a cut line to carry the whole label, even when it couldn't see the op word. Those cut lines are why the first operand leans toward the anchor, and skipping them removes the lean, so later experiments pull whole lines only.

`scouting anchoring labelling preregistration`

D2.2 pivot: an operation the model has to infer

Adopted after ex-2.2.14. Since the anchor went to the op word, D2.2 now anchors an op the model infers from solved examples, and no token in the line names the op. That is closer to what M3 needs. Covers the new grammar, what could go wrong with it, and what it changes in the plan for D2.3.

Teaching the in-context task

Once the op had no word, the first question was whether an un-anchored model could learn the task at all. At first it got a little under halfway from guessing to the best possible score. Changes to training and a smaller set of ops brought it to about nine tenths of the way.

Reports

2.2.16. The in-context grammar pilot

The first experiment on the new grammar, where the model infers the op from a few solved examples. Every control got a little under halfway from guessing to the best possible score, so the pilot stopped before anything was anchored.

`pilot in-context anchoring labelling preregistration`

2.2.17. The center control plateau

More training steps, a lower learning rate, and a mask that stops each line attending to earlier ones took the control most of the way to its target, and then it leveled off. It does as well as an ideal predictor where the examples leave the op uncertain, and falls short where they settle it.

`scouting in-context training`

2.2.18. Dropping ops with similar answers

Dropping four ops whose answers often coincide with another op raised the best possible score, and the model came closer to it than any run before. Most of the gain came from dropping ops that round at random.

`scouting in-context training`

2.2.19. Training length and seeds for the seven-op set

Half the training length keeps most of the skill of the full recipe, a little short of what we asked for. A follow-up finds the model overconfident where the examples fit several ops.

`in-context training`

2.2.20. A high-rate head start before the recipe schedule

A short burst at a higher learning rate before the usual schedule kept no more of the skill than the plain run, so the plain schedule stays.

`in-context training`

Anchoring and removing the inferred op

With the control near its best, we anchored `difference` on the new grammar. The anchor settles on the example answers, where the context shows the most about the op. Removing the anchor direction everywhere takes `difference` out gradually. At first the edit spilled onto the other ops on most fully trained runs. The spill[‡] came from a stand-in for the op in the color embedding table, where there is no context yet to infer it from. With the pull kept off the first two slices and the anti-subspace weight held high there, the edit removes `difference` and leaves the other ops within the criterion on every fresh seed. That recipe, `falloff`, was the recipe of record until late October.

Spill. The side effect of an edit on the other ops.

Reports

2.2.21. The in-context grammar pilot, on the reworked recipe

We anchored `difference` on the reworked recipe. The anchor sits on the example answers, and hardly at all where the query answer is predicted. With the pull capped, so that it stops once a state is close to the anchor direction, an edit at every position takes `difference` out gradually and leaves the other ops as they were.

`pilot in-context anchoring labelling intervention`

2.2.22. Localized by depth, various pull caps, and contexts of varying length

A scout of three changes to the capped recipe: keeping the pull off some depths, other caps, and contexts with varying numbers of examples. None improved it. Keeping the pull off some depths made the edit spill onto other ops, the cap made no steady difference, and varying the number of examples cost a little skill. Pooled over contexts, the edited model answers about as an ideal predictor without `difference` would, though context by context it gets less than half of the way there.

`scout in-context anchoring intervention`

2.2.23. The slow seeds, trained for longer

A scout of the runs that miss the second rise in task skill. At 200 epochs a few anchored runs missed it, and at 400 every run made it and ended alike. The edit spills onto other ops on nearly every run that learned the HSV ops, and on none of the half-trained runs, so the clean edits of earlier experiments came mostly from half-trained runs. Training keeps every run, at 400 epochs.

`scout in-context anchoring intervention`

What the anchor follows at the example answers

A re-analysis of the stored ex-2.2.22 runs, asking why the anchor sat higher at earlier example answers for the same posterior. At an example answer, the anchor follows the posterior given that example alone, and the earlier examples barely count. So the rise of the anchor with the posterior comes from each example that fits, and it doesn't yet show the anchor holding an inferred op.

`reanalysis in-context anchoring`

Where the spill of the edit comes from

A re-analysis of the ex-2.2.23 checkpoints, editing one set of positions at a time. On most runs the edit removes `difference` and spills onto other ops at the same positions, the example answers, so no set of positions gives the removal without the spill; the other positions add spill of their own. Most anchored runs also latch one syntax embedding onto e_1, and where that token is the separator it carries most of the spill.

`reanalysis in-context anchoring intervention`

The pool temperature and anchor weight, without the embedding

A sweep of the pool temperature τ and the anchor weight λ_a on the pull that leaves out the embedding slice, one seed per trial, with smooth fits over the plane. Leaving out the embedding keeps the latch off the syntax embeddings over most of the plane, but the edit still spills about as much as on the every-slice runs without a latch, and more as the pool softens. Removal depends on τ through a sharp edge, the weight hardly matters, and the least spill sits near that edge.

`sweep in-context anchoring intervention`

Where the lean comes from

A re-analysis of the ex-2.2.23 checkpoints and the no-emb trials of the $\tau \times \lambda_a$ sweep, asking what the edit removes on the runs with no latch. Every anchored run loads lightness onto e_3 in the color embedding table, a stand-in for `difference` at the slices where nothing contextual exists, and the edit at those slices removes it from every op. Editing only blocks 2 to 4 removes `difference` with almost no spill. The anti-subspace term is far too weak per embedding to hold the table off, and λ_a scales both terms together.

`reanalysis in-context anchoring intervention`

What the anchor follows at the `=` tokens

A re-analysis of the ex-2.2.23 checkpoints and the $\tau \times \lambda_a$ sweep, reading the anchor at every `=`, where the model has to infer the op from the examples before it. On the recipe the anchor is nearly absent there. A softer pool puts it on the `=` tokens, where it sums the examples that fit `difference`, each by how well it fits, and takes no notice of the ones that do not, which the posterior would. Part of what it puts there is the same on every op.

`reanalysis in-context anchoring`

Pulling the deeper blocks only

Twelve runs with the pull on blocks 2 to 4 only and the anti-subspace term on every slice. The color embedding table leaned about a quarter as far as on the recipe, and the edit spilled less. Block 1, which nothing pulls, came to hold part of the concept and most of the remaining spill. Holding the anti-subspace weight high late in training, in place of annealing it, kept the table and block 1 nearly off e_3 and brought every run within the selectivity criterion, though removal fell somewhat.

`scout in-context anchoring intervention`

Setting the terms by slice

Holding the anti-subspace weight high on the embedding and block 1 only, and low on the pulled blocks, kept most of the selectivity of a high hold everywhere and gave back the removal it cost. Pooling the pull over slices as well as positions settled it on the last block on most seeds, but on one seed it settled on the query answer, a position the prediction does not use, and there the op could not be removed.

`scout in-context anchoring intervention`

The split anti weight at fresh seeds

The candidate recipes from the slice-terms experiment, at eight fresh seeds. Holding the anti-subspace weight high on the first slices only kept the op removable and the edit selective, and a weight that falls off linearly with depth did as well, within the criterion on every seed. Holding it at one level on every slice lost the anchor on one seed in each arm. The linear falloff is now the recipe of record, though it missed the task gate on one op at one seed.

`preregistration in-context anchoring intervention`

What the edit does, and how we measure it

With `falloff` in hand, we looked at what the edit does beyond removal and spill, and at the baseline the edit is measured against. The edit stays within the bound the geometry sets, and lands about half of the way to the target null. The anchor does not rely on e_3 being a coordinate axis, but the comparison with the control did, so the baseline is now the median over the other axes of the run.

Reports

What the edit does on the recipe of record

A re-analysis of the stored `falloff` runs. The edit turns the states of the other ops no further than the clean geometry allows, and the loss on each `difference` context follows how much it needed the op. The edited answers go to the other ops in about the right overall proportions, but context by context they weigh them differently from the target null, and that is most of the distance left.

`reanalysis in-context intervention grading`

Anchoring on an axis the optimizer does not favor

Adam makes the coordinate axes of the residual stream special, and e_1 is one of them. With the basis of the optimizer rotated, so that e_1 is an ordinary direction, the anchor held and the edit removed and spilled as before. Only the comparison with the control suffered, where e_1 happened to overlap the syntax direction of the control.

`scout in-context anchoring methodology`

Optimizers with no privileged basis

The control under three optimizers that treat every direction alike. The coordinate axes became ordinary directions, but the stream was no more even for it: the syntax tokens still get a direction of their own. Muon learned the task as well as Adam once its learning rate was raised.

`scout in-context training`

A baseline from the quiet axes

Removal and spill had been measured net of the same edit on the control, so they depended on where e_1 sat in the control. A re-analysis against the median drop over edits along the other coordinate axes of the run itself. The numbers barely move, and the short seeds of the rotated-basis runs come back beside the rest, so this median is now the baseline.

`reanalysis in-context intervention methodology`

Why editing block 1 alone removes the op

A re-analysis of the stored `falloff` and `τ_f fixed` runs, following the alignment[*] through the model one sub-step at a time. Block 1 is the first block that can judge an example against its operands, and on both recipes it writes about half of the evidence for the op onto e_1, whether or not it is pulled; the later blocks mostly add to what arrives. One later block reads the example answers from the query `=`, usually block 3, so an edit removes the op when it sits between block 1 and that reader. That accounts for the seeds where the edit edits at blocks 2 to 4 fell short.

Alignment. How far a run's points sitting in e_1.

`reanalysis in-context anchoring intervention`

Does the edit miss the mid-block state?

Inside each block the MLP reads what the attention just wrote, before any edit between blocks acts. A re-analysis of the stored `τ_f fixed` runs with the e_1 edit at those mid-block states too. It removes a little less of the op and spills on half the runs, since the mid-block state has e_1 on every op, so the edit between blocks stays.

`reanalysis in-context intervention`

A recipe that keeps the anchor on every run

`falloff` lost the anchor on about one run in forty, and its close variants on more. A pull pooled over blocks 1 to 3, with a temperature that starts soft, kept the anchor on every run but spilled more. Sharpening the pool over slices took the spill away, and that recipe, `τ_f fixed`, kept the anchor on every run at fresh seeds and is now the recipe of record.

Reports

An anchor weight that ramps with depth

A pull weight that rises in a straight line with depth, in place of the step at slice 2, to mirror the falloff of the anti weight. With the slice-4 weight of the step, the ramps lost the anchor on two or three of eight seeds, where the step lost none. Scaled up to the total of the step, the ramp from the embedding kept it on every seed but spilled a little more. The losses split off during the pull warm-up and followed how little pull the earlier slices got, so the step stays.

`scout in-context anchoring intervention`

A soft pool while the anchor settles

A pooling temperature[†] that starts soft and sharpens over the pull warm-up, crossed with a pull on slices 1 to 3 only. Every arm with the soft start kept the anchor on every seed, but each spilled more than its twin at the fixed temperature, perhaps because the other ops come partway onto e_1 while the pool is soft. The ramp at the fixed temperature lost the anchor on one seed in a new way: the alignment came back late, and the removal did not. The step and the fixed temperature stay.

Pooling temperature. How sharply the anchor pull picks its positions.

`scout in-context anchoring intervention schedules`

How often the anchor is lost

32 new seeds each of `falloff` and `step, soft τ`, stopped early and called from how far the example answers had come onto e_1. `falloff` lost the anchor on one seed of 40, and `step, soft τ` on none.

`scout in-context anchoring seed-sensitivity`

When the spill appears

The edit applied every four epochs along training of the pool over blocks 1 to 3, to see when its spill appears. On most of the runs that end over the criterion it appears early, while the pool sharpens; on one it appears late, on an op that run learned late.

`scout in-context intervention training-dynamics`

Three recipes side by side

A re-analysis of the stored runs of `falloff`, `step, soft τ` and `pool 1–3, soft τ` at the same eight seeds, under the median baseline of the D2.2 design. The soft start on the step stays close to `falloff` on every measurement. The pool over slices 1 to 3 removes about as much and costs the task least, and the evidence for the op peaks at slice 3 on every seed, but its edit spills over the criterion on two seeds. That spill enters with the edit at block 1, the one slice the pool pulls and the step leaves alone.

`reanalysis in-context anchoring intervention`

Soft pool scout

Sixteen schedules of the pool over blocks 1 to 3, screened at epoch 64. The final temperature of the pool over slices decided the early spill, and at 0.1 no run spilled early. Holding that temperature at 0.1 from the start, finished to 400 epochs, kept every anchor and removed a little more than `falloff`.

`scout in-context anchoring schedules`

Late spill scout

On the candidate from the soft pool scout, the spill that rises late in training follows the anneal of the anti-subspace weight. Holding that weight high keeps the spill off but costs removal, and sharpening the slice temperature late lowers it at no cost we could see.

`scout in-context intervention schedules`

Anneal confirmation

The pool with its slice temperature annealing slowly from 0.1 to 0.025, at fresh seeds. It kept the anchor on all eight runs and all forty of an early count, and stayed within the criteria, with one run at the spill line.

`in-context anchoring intervention seed-sensitivity`

Fixed confirmation

The pool with its slice temperature held at 0.1, paired seed by seed with the anneal. It kept the anchor on every run, removed about as much, and had no late spill. Seeds that did well under one recipe sometimes did poorly under the other, and parted early, so much of the variation between runs seems to come from how sensitive a run is to small changes. Adopted as the recipe of record.

`in-context anchoring intervention seed-sensitivity`

The recipe of record, scored for the post

With `τ_f fixed` adopted, we gave it the full set of measurements and a reference model trained without the op. The edit does on `τ_f`

`fixed` what it did on `falloff`, and a model that never learned `difference` lands only a little closer to the target null than the edited model. The last piece before the post is a comparison with directions found after training, planned in the post-hoc comparison page.

▼ **Reports**

An absolute tolerance for the write

The write tolerance of ex-2.2.1 is a ratio (a relative tolerance), so where the clean bound is a fraction of a degree a tiny turn counts as over. A re-analysis of four recipes finds an absolute tolerance in how much the bound varies between runs, about a degree. With both, the sites at symbols and separators drop out, and what stays are the operands at the first slice, on a run or so of the recipe of record and on every run of `pool 1–3, soft τ`.

`reanalysis` `intervention` `measurement`

What the edit does on `τ_ℓ fixed`

A re-analysis of the fixed confirmation's runs. The fixed confirmation scored `τ_ℓ fixed` on the full edit alone, and edit-effects scored the earlier recipe; this report gives `τ_ℓ fixed` the scoring of edit-effects, and adds removal by dose and the confusion matrices. Every measure comes out about as on `falloff`. The clean bound at slice 3 is wider, set by a small tail rich in near misses for `difference`, which fits the account of block 1 in block1-edit. For where the edited answers land, it draws on the model trained without the op rather than repeating it.

`reanalysis` `in-context` `intervention` `grading`

A model trained without `difference`

The control trained with no loss on `difference` contexts, so that it never learns the op, against the edited model of the recipe of record. It comes a little closer to the target null than the edited model, but still falls well short, and misses in the same way: the mass goes to the other ops, but it is shared among them differently from the target null on each context, and differently again at each seed.

`in-context` `intervention` `grading` `seed-sensitivity`

What the edit misses in the low runs

A few runs trained with one label draw held for the run end with the edit removing much less of `difference`, and a twin trained again from the same labels, seed, and batches matches each in loss and alignment and still removes the op well. A re-analysis of 48 stored checkpoints: every run also writes, off the anchored axis, whether each example fits the op, and in a low run the reader blocks take part of that judgement from copies the edit never touches, at the answers or at the other example positions. The low runs peak lower and give back more while the anti weight anneals.

`reanalysis` `in-context` `intervention` `seed-sensitivity`

When paired runs stop being alike

How alike two runs at one seed stay under two conditions, score by score, from the stored runs of four experiments. Started together, they keep about half of their pairing in the task score through epoch 64, and in removal and spill they are little more alike than runs at different seeds. Carried on from one stored state, they stay alike in all three for hundreds of epochs.

`reanalysis` `seeds` `methodology`

---

1. A literate script is plain Python with Markdown prose between the cells. It reads durable results produced by a separately-run experiment. Reports are published automatically, with their figures served from a Hugging Face dataset; the infrastructure is mi-ni. ↩