Ex 2.1.9: Softmin pooling over sequences

> Anchoring on op1 alone worked best so far, but in natural language we won't know which tokens hold the concept. So we pool over sequences with a soft minimum (mellowmax) and let the pull choose its own position. At the embedding it picks op1 unaided, the operating point stays healthy, and grading improves. It wins no margin though, and only the softest pooling stays graded in every run.

Findings

H1 (task cost) — holds. The largest `named_holdout` exact-match gap from the control across all five anchored conditions is 0.0065, against a gate of 0.02.

H2 (pooling increases selectivity) — unmet. On the primary condition (`pool-t030`): (a) the margin gain over the span mean is +0.034 against a gate of 0.15 (partial 0.09), and (b) the syntax-role drift falls by 0.078 against a required 0.1 — the predicted direction on both, short of both gates, and neither partial applies. One piece of context, unpacked in the selectivity section: the op1 oracle itself beats the span mean by only 0.067 on m_{span} here, so H2(a)'s gate was out of reach for any pull; `pool-t030` recovered 50% of the headroom that existed.

H3 (the operating point survives pooling) — holds. On `pool-t030`: containment $\bar{\alpha}$ = 0.045 (gate ≤ 0.1); retention 0.94 (gate ≥ 0.8, minimum across seeds); and grading r^2 = 0.84 against the span mean's 0.73 — the gate tolerated a drop of 0.1 and instead the response got better graded.

H4 (the weight lands where the concept lives) — holds. At the embedding slice `pool-t030` puts 0.77 of the softmin weight on op1 (gate ≥ 0.4; uniform is 0.25) — the only span position whose state can carry the concept there — and the weighted readability gain is +0.21 (gate ≥ 0.05). At depth the concentration is winner-take-all seed by seed: at the two sharper τ some seeds latch onto `+` in the first few epochs, and only τ = 0.1 keeps a graded profile on op1 in every seed (*Where the weight went*).

So the pull finds the concept-bearing position without the oracle, and holding the operating point costs nothing — but finding it buys almost no additional margin here. What that means for M3's document-level labels is taken up in the *Discussion*.

> **How to read this report**
>
> This report was preregistered: the method, hypotheses and decision thresholds were frozen before the experiment ran (commit `0abebb8`, with pre-launch review corrections noted in comments). Results replaced the `TODO` placeholders in place; everything conceived after seeing the data is under *Exploratory analyses*, marked as post hoc.

Ex-2.1.6 pulled the whole prompt span of *red*-labeled lines toward the anchor axis. The term was mostly satisfied by moving the whole color cube, so we got alignment without selectivity. Ex-2.1.7 found that two fixes stack: **(a)** a repulsive anti-subspace term, and **(b)** narrowing the pull to op1 alone. Ex-2.1.8 tuned the schedule of the repulsion (a) and landed on an operating point (`end90-hold30`) where the span pull holds the cube near the control ($\bar{\alpha}$ = 0.09) while keeping the margin (0.61).

That leaves the narrowing fix (b), which we can't keep. Pulling only op1 works because this language has exactly one position that names the labeled concept, and we know which one it is, whereas a document-level label on natural text gives us a span and nothing else. Can the pull *find* the right position on its own, given the freedom to spread its budget unevenly?

The pooled term (below) asks only that each labeled line align *somewhere* in its span, so the gradient concentrates on whichever position is already cheapest to align. The cheapest position is a shared syntax token: `+` and `=` appear in every line, so one nudge to their embedding satisfies every labeled line at once.

The anti-subspace term should make that option unattractive. Every line, labeled or not, feels repulsion on a syntax token that sits on the axis, while repulsion on an alignment *specific to red operands* is countered by the pull where red actually occurs. If that balances out, the pull should settle on op1 at the embedding, the only span position whose state varies with the labeled color, and then follow the concept as it migrates across positions with depth, which ex-2.1.6 saw the un-anchored model do.

It might not work out. The ex-2.1.8 span pull actually aligned the syntax tokens the most: at the embedding its best-aligned span roles (positions named by their job in the line: op1, `+`, op2, `=`) are `+` and `=`, and the cube-wide drift is there too. ᾱ measured at op1 is 0.09, but the same statistic maxed over the span roles is 0.37.[1] The pooled term would happily settle on that. Call that outcome the *syntax latch* — a latch because the softmin's concentration is self-reinforcing (the leading position gets the pull, which extends its lead), so whatever it closes on first, it holds. The question is whether the repulsion redirects it before it closes.

Method

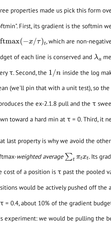
Everything follows ex-2.1.8 except the pooling: the word-level `v216` corpus from ex-2.1.5 (216 colors, one token each, d64-L4), the same seeds, sequence-level labels, measurements, anchor schedule, and the anti-subspace schedule fixed at the operating point of that experiment (`end90-hold30`) in every anchored cell.

The pooled term

Previously, the anchor term averaged $x_t = 1 - \cos(h_t, \hat{v})$ over every pulled position. We replace that mean, per labeled line, with a soft minimum over the span of the line, the *mellowmax*[5]

$$\mathrm{mm}_\tau(x) = -\tau \log \frac{1}{n} \sum_{t \in \text{span}} e^{-x_t/\tau},$$

where τ is a "temperature" that shapes how the pull distributes over the span: an even spread at large τ, concentrating on the cheapest positions as τ falls. The per-line pooled values are averaged over labeled lines and residual-stream slices as before.[2]



Pooling happens within each slice, so the pull can choose different positions at different depths, following the concept as it migrates.

Three properties made us pick this form over the other things called "softmin". First, its gradient is the softmin weights $\pi_t = \mathrm{softmax}(-x/\tau)_t$, which are non-negative and sum to 1, so the pull budget of each line is conserved and λ_a means the same thing at every τ. Second, the $1/n$ inside the log makes $\tau = \infty$ the span mean (we'll pin that with a unit test), so the $\tau = \infty$ condition reproduces the ex-2.1.8 pull and the τ sweep interpolates from there down toward a hard min at $\tau = 0$. Third, it never pushes.

That last property is why we avoid the other common "softmin", the softmax-*weighted average* $\sum_t \pi_t x_t$. Its gradient terms flip sign once the cost of a position is τ past the pooled value, so poorly-aligned positions would be actively pushed off the axis (on a two-position toy at $\tau = 0.4$, about 10% of the gradient budget). That would confound this experiment: we would be pulling the best position and pushing the rest.

A training crop can truncate a line, so the pool runs over the *visible* span positions of the line, and each labeled line counts once. At $\tau = \infty$ that's a per-line mean of means rather than the flat mean over positions used in ex-2.1.8. The two differ only on truncated lines, and the reproduction check below will say whether that (or anything else in the new code path) moved the numbers.

Conditions

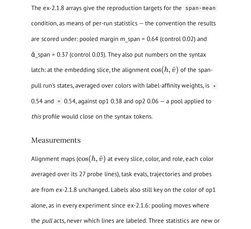
Six conditions, three seeds each. Every anchored cell runs the anti-subspace term at the ex-2.1.8 operating point; only the pooling varies.

condition	λ_a	τ	pulled positions
`lam0`	0	∞	prompt span
`span-mean`	0.1	∞	prompt span
`pool-t010`	0.1	0.01	prompt span
`pool-t030`	0.1	0.03	prompt span
`pool-t100`	0.1	0.1	prompt span
`op1-oracle`	0.1	∞	op1 only

`span-mean` re-runs the `end90-hold30` condition of ex-2.1.8 through the pooled code path at $\tau = \infty$: the baseline the pooled conditions have to beat, differing from them only in τ. `op1-oracle` is the op1-only pull of ex-2.1.7 at the new operating point: what knowing the right position buys, measured through identical code, so we can say what fraction of it pooling recovers. It is a reference rather than a strict ceiling — the oracle pulls op1 at every slice, and ex-2.1.6 saw the concept migrate to later roles with depth, so a per-slice pool could in principle beat it.

Calibrating τ

The interesting τ range depends on how much x_t varies across the span, which we can read off the stored alignment maps of ex-2.1.8 before running anything. The figure below computes what the softmin weights *would* be on two reference profiles: a fresh run (the un-anchored control) and a trained one (the span pull at the operating point).



***Calibrating the τ grid from the ex-2.1.8 stored alignments.** The leading span position's share of the softmin weight, by residual slice, if the weights were computed from the alignment profile of the un-anchored control (**left**) or the span pull at the operating point (**right**). The shaded band is the 0.6–0.8 target from the design note; dashed verticals are the chosen grid. Both profiles put the band at $\tau = 0.01$–0.03, so the grid brackets it with one rung toward the mean.*

The design note targets a leading position taking 0.6–0.8 of the weight, which lands at $\tau = 0.01$–0.03 on both profiles.

So the grid is $\tau \in \{0.01, 0.03, 0.1\}$: the sharp edge of the band, the soft edge, and one rung toward the mean and end of the τ axis ($\tau = 0.1$, weights nearer uniform); the mean itself is the $\tau = \infty$ condition, which closes the family. $\tau = 0.01$ sits below the per-seed margin noise (0.056), so at that rung the pool will commit hard to early differences that may just be noise. That is the winner-take-most end of the sweep.

The ex-2.1.8 arrays give the reproduction targets for the `span-mean` condition, as means of per-run statistics — the convention the results are scored under: pooled margin $m\_span = 0.64$ (control 0.02) and $\hat{a}\_span = 0.37$ (control 0.01). They also put numbers on the syntax latch: at the embedding slice, the alignment $\cos(h, \hat{v})$ of the span-pull run's states, averaged over colors with label-affinity weights, is $+0.54$ and -0.54, against op1 0.58 and op2 0.06 — a pool applied to this profile would close on the syntax tokens.

Measurements

Alignment maps $(\cos(h, \hat{v})$ at every slice, color, and role, each color averaged over its 27 probe lines), task evals, trajectories and probes are from ex-2.1.8 unchanged. Labels also still key on the color of op1 alone, as in every experiment since ex-2.1.6: pooling moves where the *pull* acts, never which lines are labeled. Three statistics are new or newly scored, all computed from the same maps:

The pooled margin `m_span` is the mean over slices of the max-over-span margin: for each slice, take the margin at each span role and keep the largest, then average over slices. The margin at a role is selectivity rather than raw alignment: over the 216 colors, the affinity-weighted mean alignment of their states at that (slice, role), minus the unweighted mean — how much closer a line sits to the axis for having a red op1. At depth the syntax roles' states vary with the color of op1 too (they attend back to it), so the margin means the same thing at every role; there is no separate notion of how aligned a role *should* be. We can't score the margin at op1 alone, because one condition pulls exactly where we would be scoring, and op1 may not be where the concept belongs at depth anyway.

The max is applied identically to every condition and to the control, so the maximum-of-noise inflation it invites cancels out in the control-subtracted comparisons the hypotheses make. We report m_{op1} beside it for continuity with ex-2.1.7 and ex-2.1.8.

`â_span` is the counterpart for containment, with the same max treatment: for each slice, the unweighted mean alignment over the 216 colors' states at each span role, keeping the largest role; then the mean over slices. Color enters only through that mean — this is drift of the whole cloud, not selectivity. The usual â reads at op1 only, which the ex-2.1.8 operating point shows isn't enough here: its cube-wide drift sits on the syntax roles, where the op1 statistic can't see it.

The softmin weight profile $\pi(l, t)$ records what the pull actually chose. (It, since `sr` collides with model weights.) At the end of training we compute the weights $\mathrm{softmax}(-x/\tau)$ on the alignment probe set (every color as op1 against all 27 closed partners), per slice and span role, weighted by label affinity over colors.

Its reference is **the readability profile of the control** $R^2(l, t)$: a ridge probe predicting the redness of op1 from the state of the *un-anchored control* at each (slice, role), with one op1 color held out at a time. This R^2 is the probe's held-out coefficient of determination (negative on a role that carries nothing), a different quantity from the squared Pearson correlation that grading uses; earlier reports wrote both as R^2, so here they are R^2 (probes) and r^2 (grading). At the embedding we know this profile in advance, since op1 is the only span position whose state varies with the color of op1. H4(a) relies on this for its prediction.

Hypotheses

Gates and noise floors are from ex-2.1.6 through ex-2.1.8 wherever the statistic is unchanged. Derivations for the new gates are in experiment.py, next to their constants.

In the hypothesis statements, *P* names the primary pooled condition, fixed in advance as `pool-t030`: on both reference profiles its τ sits at the calibration band's soft edge on the shallow slices, and below the band on the deeper ones — the conservative end of the target range — a criterion independent of every gated statistic. H2–H4 all score P; the results prose calls it `pool-t030` or "the primary condition". The other two pooled conditions are the τ sweep, reported beside P but not gated. If P turns out task-dirty, H2–H4 go unscored and the refuted H1 is the finding.

H1: Task cost. Every condition is task-clean: `named_holdout` exact match within 0.02 (absolute) of the seed mean of the in-experiment control. No partial credit: ex-2.1.7 found no task cost even at ten times this anchor weight, and nothing in this design should add task pressure.

H2: Pooling increases selectivity over the span mean. (a) $m_{\text{span}}(P) - m_{\text{span}}(\text{span-mean}) \geq 0.15$, about two thirds of what the op1 oracle gave at op1 in ex-2.1.7 (0.22), and roughly 3× the noise of a difference of three-seed means. (b) The syntax-role drift falls: $\bar{\alpha}_{\text{span}}(P) \leq \bar{\alpha}_{\text{span}}(\text{span-mean}) - 0.1$. Partial: (b) holds and (a) lands in $[0.09, 0.15)$; or exactly one of (a), (b) holds outright. (b) is a prediction of the mechanism: a pull that concentrates its budget should stop spending alignment on positions that don't help it. A condition that gains margin while leaving the syntax drift in place has found some other mechanism.

H3: The operating point survives pooling. On P: (a) containment, $\bar{\alpha} \leq 0.1$ at op1; (b) retention, every run whose running maximum of m_{span} reaches 0.2 ending at $\geq 0.8\times$ that maximum (quoted as the minimum across seeds); (c) grading, the squared Pearson correlation r^2 against `sim^1.5` of P's seed-mean op1 response no more than 0.1 below the `span-mean` condition's (higher is fine).[4] Partial: (a) and (b) hold, (c) misses.

H4: The weight lands where the concept lives. On the weight profile of P: (a) at the embedding slice, the leading role is op1 with weight ≥ 0.4 (uniform is 0.25; this gates the *trained* weights, where the 0.6–0.8 calibration band described weights recomputed on reference profiles); (b) across the four slices after the embedding, the weighted readability gain $-\sum_t \bar{\pi}(l,t)R^2(l,t) - \text{mean}_t R^2(l,t)$, averaged over slices — is at least 0.05. Partial: (a) holds and (b) lands in $[0, 0.05)$. Two contrary outcomes, both leading to the same conclusion: span labels need help from the label side (per-line reweighting), not just the loss side. *Latch*: the syntax latch from the intro arrives — the embedding weight concentrates on `+` or `=`, meaning the pull took the cheap shared tokens and the repulsion failed to make them unattractive; a latch would take H2(b) down with it — one mechanism, seen from the loss side there and the weight side here. *Uniform*: no slice reaches 0.4 in any pooled condition, meaning the pool can't tell the positions apart at any τ short of collapse.

Task cost (H1)

condition	seen EM ↑	holdout EM ↑	holdout NLL ↓	`open` dist ↓	Δ holdout EM	H1 gate
`lam0`	1.000 ±0.000	1.000 ±0.000	0.016 ±0.005	0.138 ±0.005	+0.0000	clean
`span-mean`	1.000 ±0.000	0.997 ±0.002	0.019 ±0.006	0.151 ±0.016	-0.0026	clean
`pool-t010`	1.000 ±0.000	0.996 ±0.006	0.023 ±0.009	0.132 ±0.002	-0.0039	clean
`pool-t030`	1.000 ±0.000	0.996 ±0.004	0.028 ±0.010	0.137 ±0.006	-0.0039	clean
`pool-t100`	1.000 ±0.000	1.000 ±0.000	0.014 ±0.002	0.135 ±0.007	+0.0000	clean
`op1-oracle`	1.000 ±0.000	0.993 ±0.006	0.022 ±0.009	0.133 ±0.001	-0.0065	clean

Behavior by condition. Each value is the seed mean, with half the seed range beside it. EM is exact match on the answer token, NLL its surprise in nats, and `open` dist the RGB distance from the guessed color to the true mix on pairs with no named answer. Δ holdout EM is the signed gap to the in-experiment control, which the H1 gate scores at 0.02.

H1 holds. The largest gap from the control is 0.0065, at `op1-oracle`, against a gate of 0.02. Every condition solves the task, so H2–H4 are all scored. Concentrating the whole pull budget of a line onto one position — which at $\tau = 0.01$ is nearly a hard min — costs nothing the task loss can see, consistent with every anchored experiment on this testbed so far.

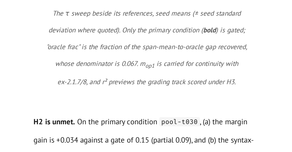
Selectivity against the span mean (H2)

First the reproduction check the method promises, so any drift introduced by the new code path is visible before a pooled condition is read: the `span-mean` condition re-runs ex-2.1.8's operating point through the pooled code at $\tau = \infty$, so it should land on that experiment's numbers.

condition	m_{span}	ex-2.1.8	Δ	$\bar{\alpha}_{\text{span}}$	ex-2.1.8	Δ	m_{op1}	ex-2.1.8	Δ	$\bar{\alpha}$ (op1)	ex-2.1.8	Δ
span-mean	0.648 ±0.014	0.640	+0.008	0.555 ±0.006	0.570	−0.015	0.619 ±0.019	0.610	+0.009	0.100 ±0.005	0.087	+0.013
Lam0	0.023 ±0.016	0.015	+0.008	0.018 ±0.017	0.054	−0.036	−0.029 ±0.025	−0.050	+0.001	−0.012 ±0.020	0.008	−0.020

The reproduction check. Each statistic is this experiment's seed mean (± seed standard deviation) beside the same statistic recomputed from ex-2.1.8's published arrays under the same per-run convention, and the difference. `span-mean` re-runs `end98-ho1d38` through the pooled code path; `Lam0` re-runs its control.

The largest differences — +0.008 on m_{span} and -0.015 on $\bar{\alpha}_{\text{span}}$ — sit within a seed spread of the targets, and the same is true of the control. So the per-line mean-of-means (which differs from ex-2.1.8's flat mean only on crop-truncated lines) reproduces the operating point, and the pooled conditions can be read against `span-mean` as a like-for-like baseline.



***The H2 statistics, control-subtracted.** Left: m_{span}; right: $\bar{\alpha}_{\text{span}}$. Rows run from the span mean ($\tau = \infty$) down the sweep to the oracle. Small dots are seeds, the tick their mean; the $\tau = \infty$ and oracle reference rows are greyed. Dashed carets mark the H2 gates for P: a margin gain of 0.15 over `span-mean` (dotted: the 0.09 partial) and a drift drop of 0.1 below it. The two panels share their scale, so the sizes of margin and drift compare directly.*

condition	m_{span} ↑	− control	− span-mean	oracle frac	$\bar{\alpha}_{\text{span}}$ ↓	m_{op1}	r^2
span-mean	0.648 ±0.014	0.625	+0.000	—	0.555 ±0.006	0.619	0.73
pool-t010	0.651 ±0.019	0.628	+0.003	5%	0.510 ±0.006	0.585	0.77
pool-t030	0.682 ±0.015	0.659	+0.034	50%	0.276 ±0.047	0.648	0.84
pool-t100	0.704 ±0.017	0.681	+0.056	83%	0.242 ±0.018	0.704	0.83
op1-oracle	0.715 ±0.016	0.692	+0.067	—	0.147 ±0.005	0.715	0.87

*The τ sweep beside its references, seed means (± seed standard deviation where quoted). Only the primary condition (**bold**) is gated; 'oracle frac' is the fraction of the span-mean-to-oracle gap recovered, whose denominator is 0.067. m_{op1} is carried for continuity with ex-2.1.7/8, and r^2 previews the grading track scored under H5.*

H2 is unmet. On the primary condition `pool-t030`, (a) the margin gain is +0.034 against a gate of 0.15 (partial 0.09), and (b) the syntax-role drift falls by 0.078 against a required 0.1. Both moved the way the mechanism predicts and neither reaches its gate, so neither partial applies.

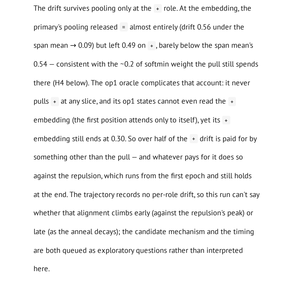
Three things the gate arithmetic hides:

1. **The margin headroom collapsed before the sweep began.** The gate was sized from the op1 oracle's +0.22 margin gain in ex-2.1.7 — measured at op1, under the old schedule. Here, scored as m_{span}, at the ex-2.1.8 operating point, the oracle's whole advantage over the span mean is 0.067: less than half the gate. No condition, oracle included, could have passed H2(a) as preregistered. The primary recovered 50% of the headroom that existed.

2. **Neither statistic is monotone in τ.** In the dot chart both panels walk from `span-mean` toward the oracle only as far as τ = 0.1, then step back: the sharper the pool, the closer the condition sits to the span mean again. What the statistics do track is op1's share of the pull budget — 0.25 for the span mean by construction, then 0.90 at τ = 0.1, 0.68 at τ = 0.03, 0.49 at τ = 0.01 (mean over slices of the weight profiles under H4), and 1.0 for the oracle. The share peaks at τ = 0.1 because a sharper pool concentrates the budget harder but risks putting it on the wrong role: at depth, each seed of the two sharper rungs commits everything to a single role, and at τ = 0.01 two seeds of three commit to `+` (at τ = 0.03, one; at τ = 0.1, none). The same decomposition holds within a condition — `pool-t030`'s committed-to-`+` seed (s0) scores m_{span} 0.62 and $\bar{\alpha}_{\text{span}}$ 0.35 where its op1-keeping seeds score ≈0.71 and ≈0.24. So the finite-τ means differ mostly through how many seeds latched, and the sweep's message is a latch-rate trend rather than a smooth dose response; with three seeds per rung the rates themselves are coarse, so carrying a τ forward should rest on the latch mechanism (*Where the weight went*), with more seeds at the chosen rung in the next design.

3. **Why the span mean scores near the oracle.** Near on the absolute scale: its m_{span} is 91% of the oracle's, where the un-anchored control sits at 0.02 — the whole sweep lives in the table's last few hundredths. The role profiles below show why: at depth, every span role of the `span-mean` condition carries margin (its syntax-role states attend back to op1, so they become selective too), and a max over roles only needs one good role. Pooling *did* change the shape — margin and drift both withdraw from op2 and `=` almost entirely — but op1's own margin is nearly the same in every anchored condition, so the max barely moves.

***Where the margin and the drift live.** Margin (top) and drift (bottom) by span role, per slice, seed means: $m(\ell, t)$ and the unweighted mean alignment $\bar{\alpha}(\ell, t)$. Columns: the span mean, the primary, and the op1 oracle. The scored statistics take the per-slice max over these lines (m_{span} from the top row, $\bar{\alpha}_{\text{span}}$ from the bottom), so a single high line is all either statistic sees.*

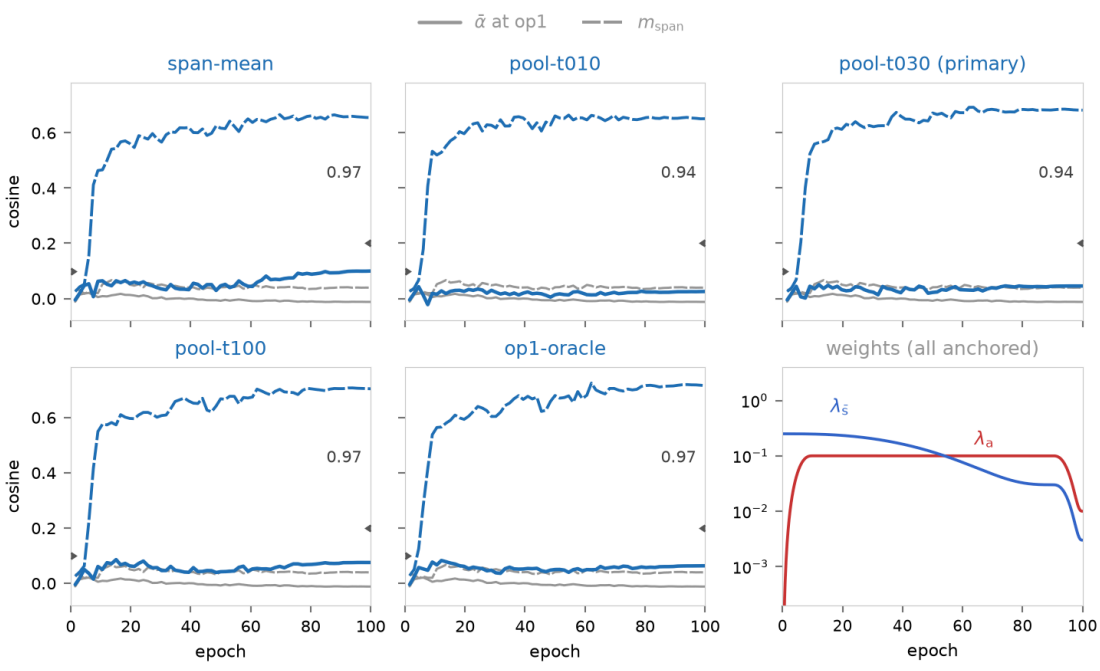


***The same profiles, position-major.** Margin $m(\ell, t)$ (shaded area) and drift $\bar{\alpha}(\ell, t)$ (line) over the four span roles, one panel per residual slice with the embedding at the bottom — the transpose of the figure above, in the layout the weight profiles under H4 use. Columns: the span mean, the primary, and the op1 oracle.*

The drift survives pooling only at the `+` role. At the embedding, the primary's pooling released `=` almost entirely (drift 0.56 under the span mean → 0.09) but left 0.49 on `+`, barely below the span mean's 0.54 — consistent with the −0.2 of softmin weight the pull still spends there (H4 below). The op1 oracle complicates that account: it never pulls `+` at any slice, and its op1 states cannot even read the `+` embedding (the first position attends only to itself), yet its `+` embedding still ends at 0.30. So over half of the `+` drift is paid for by something other than the pull — and whatever pays for it does so against the repulsion, which runs from the first epoch and still holds at the end. The trajectory records no per-role drift, so this run can't say whether that alignment climbs early (against the repulsion's peak) or late (as the anneal decays); the candidate mechanism and the timing are both queued as exploratory questions rather than interpreted here.

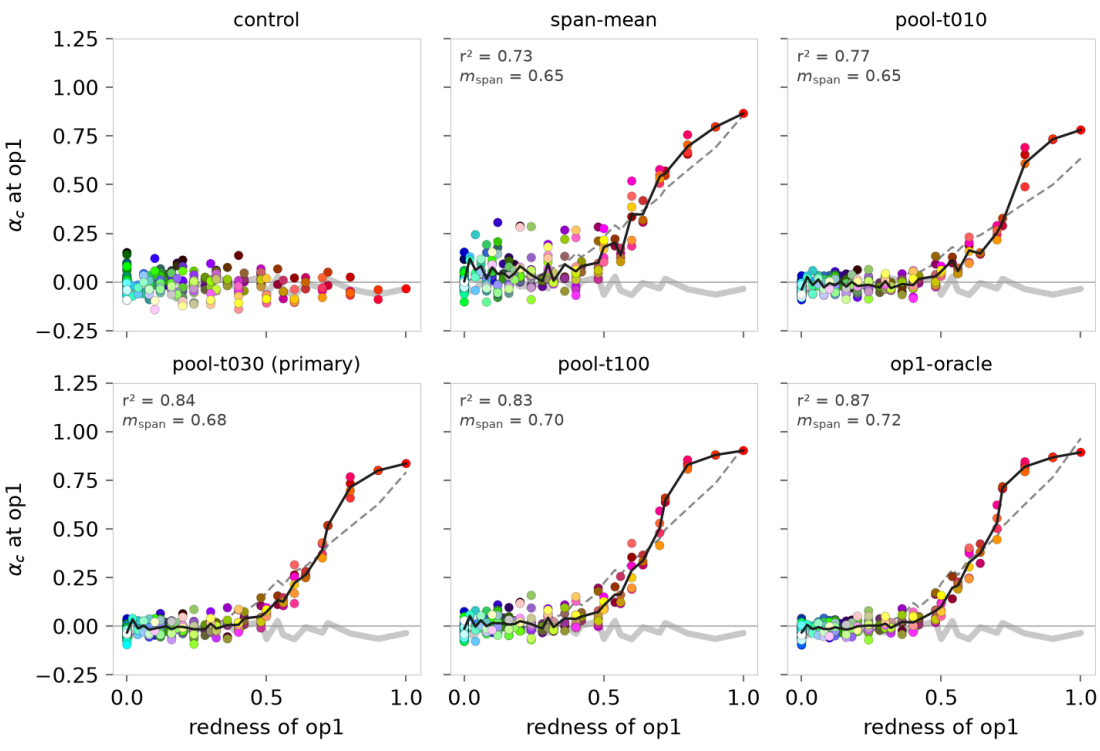
The operating point under pooling (H3)

The endpoint numbers say where each run finished; the trajectories say how it got there — in particular whether a pull that concentrated early slid once the repulsion annealed away, which was the contrary outcome H3 named.



Training dynamics by condition. *Seed means of the two cosines on the anchor axis, on one shared scale: solid is $\bar{\alpha}$ at op1 (contained at the left caret, 0.1); dashed is m_{span} (its retention floor of 0.2 at the right caret). The grey pair is the un-anchored control, and the number in each panel is that condition's retention (final over peak m_{span}, minimum across seeds; gate ≥ 0.8). Bottom right: the weight schedule every anchored condition trained under — anchor λ_a in red, repulsion $\lambda_{\bar{s}}$ in blue, log scale — shared here because the anti-subspace term is fixed at the ex-2.1.8 operating point in every anchored condition.*

H3(a) and (b) hold. On the primary condition, $\bar{\alpha}$ at op1 ends at 0.045 ±0.037 against the 0.1 gate, and retention is 0.94 against 0.8. The contrary outcome — a concentrated pull sliding once the anti anneal ends at epoch 90 — shows up only as the small end-of-training dip every condition shares (the anchor's own anneal), and the worst retention anywhere in the sweep is 0.94. Pooling redirects the budget of the pull without destabilizing what ex-2.1.8 found.

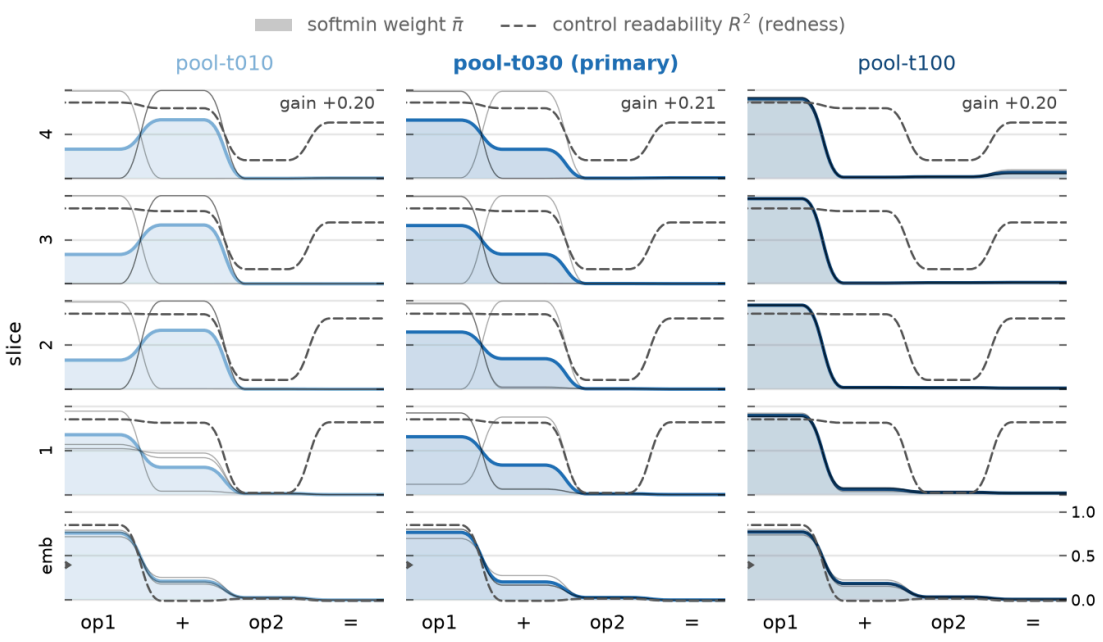


Grading: alignment at op1, per color. *α_c, the mean over slices of $\cos(h, \hat{v}_{\text{red}})$ at op1, seed mean, against the redness of the color. One mark per color, drawn in that color; the thin dark line is the mean response at each of the 29 distinct redness levels, the flat grey band the same line for the control, and the dashed line `sim^1.5` rescaled onto the response by least squares — the shape the r^2 statistic scores against. The dashed line wobbles because `sim` depends on the full color (hue angle, weighted by vibrancy), so its mean at a redness level moves with the mix of colors that share the level. H3(c) gates the primary's r^2 relative to `span-mean`'s.*

H3(c) holds, with the sign reversed. The gate would have tolerated r^2 on the primary falling 0.1 below the span mean's; instead it lands +0.10 *above* it (0.84 against 0.73). The contrary expectation — that concentrating the pull would sharpen the response into a step — mostly has it backwards: every pooled condition grades at or above the span mean, and the oracle best of all (0.87). Withdrawing the pull from positions that don't carry the concept apparently cleans the response shape at the one that does. The exception is visible in the `pool-t010` panel: a shelf in the rise near redness 0.8, which turns out to belong to the latched seeds (*Exploratory analyses*). As a footnote to ex-2.1.8's open question about where a realistic absolute grading bar sits: the primary and the oracle both clear the 0.8 that experiment couldn't reach.

Where the weight went (H4)

The weight profile is the pull's own account of where each line's budget goes: the softmin weights, recomputed after training from the final model's alignments on the probe set — the same formula the loss's gradient uses, so this is what the pull was spending as training ended (the timing section below reads the logged version over training). The control's readability profile is the reference for where the concept actually lives. H4 asks whether they agree — at the embedding, where only op1 can carry the concept, and across depth, where the syntax roles pick it up through attention.



What the pull chose, against where the concept is readable. *The trained softmin weight profile of each pooled condition (shaded area, seed mean $\bar{\pi}(\ell, t)$; hairlines, the three seeds individually), over the four span roles, one panel per residual slice with the embedding at the bottom. Where the hairlines separate — the deep slices at τ ≤ 0.03 — each seed's profile is one-hot on a single role, and the mean is a mixture of seeds rather than a shape any run has. The dashed line is the readability profile of the un-anchored control, $R^2(\ell, t)$ — a ridge probe predicting op1's redness from the state at each (slice, role), one color held out at a time — the same reference in every column of a row (its own seed spread is ≤ 0.09 everywhere except op2 at depth, ≈ 0.2). The caret at the embedding marks the H4(a) gate: op1's weight ≥ 0.4 (uniform is 0.25). The number atop each column is its H4(b) weighted readability gain (gate ≥ 0.05, scored on the primary).*

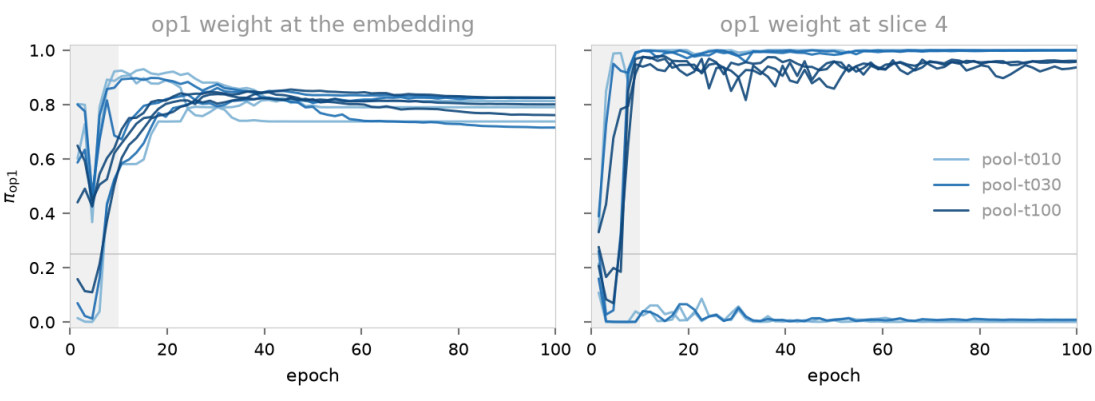
H4(a) holds. At the embedding, the primary's leading role is op1 with weight 0.77 against the 0.4 gate — and the same is true at every τ (0.76 at 0.01, 0.77 at 0.1). Neither contrary outcome occurred where the gate reads: no condition latched the embedding onto `+` or `=` (readability ≈ 0 there), and none stayed uniform. The trained weights concentrate harder than the 0.6–0.8 calibration band predicted from the reference profiles, which is the softmin's self-reinforcement doing what the introduction said it would — just in the right place.

H4(b) holds. The primary's weighted readability gain is +0.21 against the 0.05 gate (pool-t010 +0.20, pool-t100 +0.20). Most of the gain comes from avoiding op2 — the one span role that stays hard to read at depth (R^2 0.12 averaged over the four deep slices, against 0.81 for the other three roles).

Across depth the conditions diverge, and seed by seed the divergence is all-or-nothing: the hairlines show each run of the two sharper rungs committing its entire deep-slice weight to a single role. At τ = 0.01 that role is `+` in 2 seeds of three and op1 in the other; at τ = 0.03, `+` in 1; at τ = 0.1 none commit to `+`, and the profile is the sweep's only genuinely graded one at depth (op1 at 0.87–0.97 per seed, with real weight left elsewhere). So the seed-mean's apparent `+` minority at τ = 0.03 is a mixture of one-hot seeds, and the τ trend is a latch-rate trend: 2/3 → 1/3 → 0/3. A `+` latch is a cheap early choice rather than a ruinous one, since `+` at depth reads op1's redness at $R^2 \approx 0.82$. The prereg expected migration *toward* the syntax roles with depth in step with readability; what actually distinguishes the healthy runs is that they keep op1 — which stays the *most* readable role at every slice — and the pull only leaves it where τ is sharp enough to lock in an early accident.

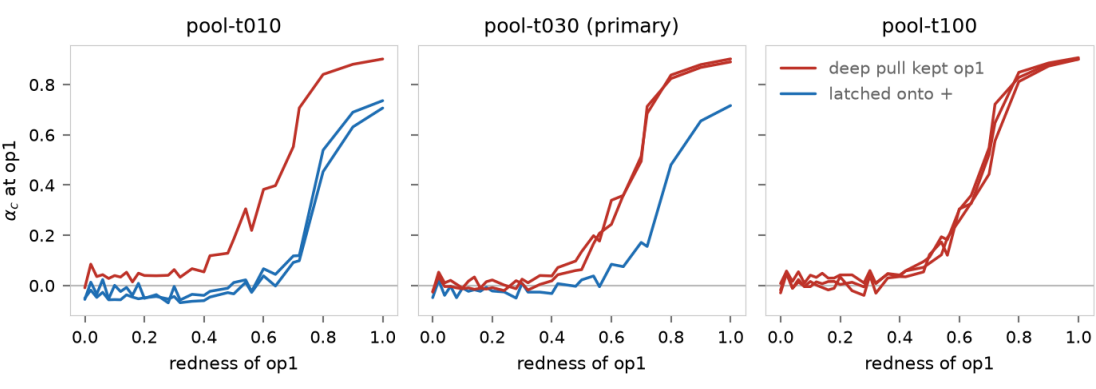
Exploratory analyses

Nothing here is preregistered; everything in this section is post hoc.



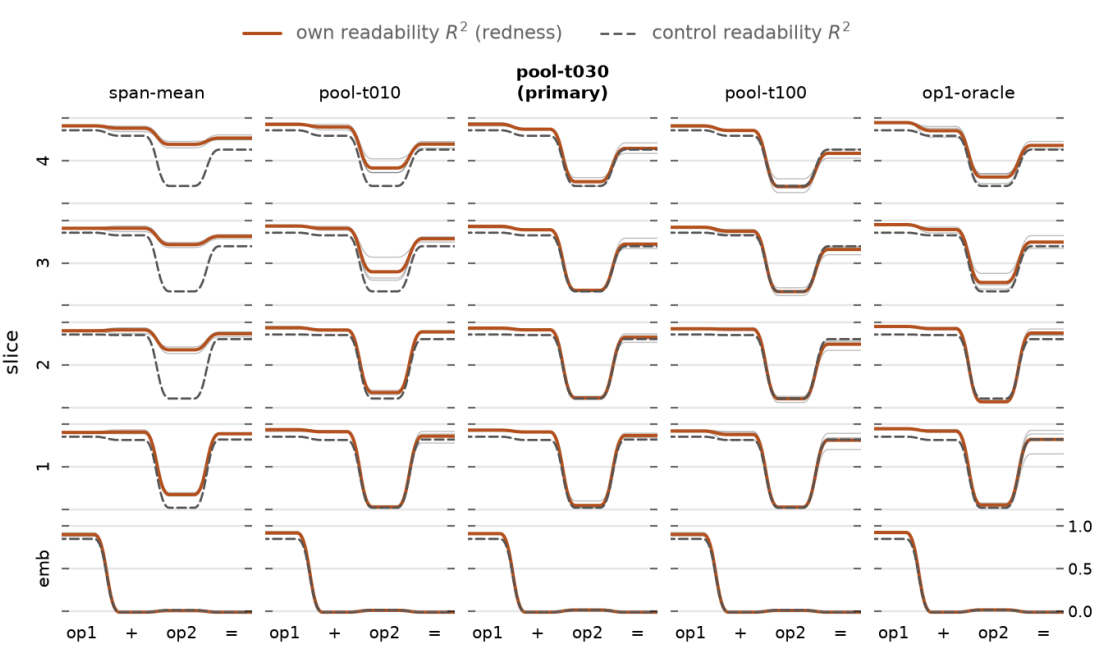
When the pull decides. *op1's share of the softmin weight over training, at the embedding (**left**) and at the last slice (**right**), one line per run; the thin rule is uniform (0.25). The vertical band is the anchor's warmup window (epochs 0–10); the repulsion anneal runs to epoch 90.*

The choice is made during the anchor ramp, run by run. Every run's embedding weight settles by about epoch 20, and every run's deep-slice choice is committed by epoch 8 — the τ = 0.01 latches by epoch 2–3, before the anchor has even finished warming up — and no choice changes for the rest of training. Nor is there much reason it should: once the softmin has concentrated, the losing roles receive no pull, and late training supplies no perturbation of the needed size — the learning rate is decaying, the regularizer weights are annealing, and 100 epochs is generous for this corpus, so the task loss has long converged. Two consequences: the long repulsion anneal that ex-2.1.8 tuned matters for containment rather than for position choice (whatever the repulsion contributes to the choice, it contributes in the first few epochs, before the softmin closes), and any intervention on the choice — a τ schedule that starts soft and sharpens, say — has to land in that same window. The latch, then, is a race at initialization: whichever role happens to align cheapest in the first few hundred steps keeps the weight, which is why the winner-take-most rung latches most often.



The shelf, split by each run's deep-slice choice. *The grading response of every pooled run — the mean response at each of the 29 redness levels, as in the H3(c) figure but one line per run — colored by where that run's slice-4 softmin weight ended: op1 kept (red) or latched onto `+` (blue).*

The shelf belongs to the latched runs. In the runs whose deep slices kept op1, the response grades smoothly with redness at every τ. In the runs that latched onto `+`, it holds up only at the red end and drops sharply below redness 0.8 — `pool-t010`'s two latched runs average 0.50 at redness 0.8 and 0.11 at 0.72, where its op1-keeping run reads 0.71 — which is also where the label affinity falls away steeply. That reading makes sense of the mechanism: in a latched run only the embedding slice still pulls op1, so a color keeps its alignment only to the extent that its lines keep getting labeled, and the graded contribution the deep slices add in healthy runs is missing. The seed-mean scatter under H3(c) blends the two shapes, which is what the `pool-t010` panel's shelf is.



Where redness is readable, once a pull has acted. *The same leave-one-color-out ridge probe as H4(b)'s reference, run on the anchored checkpoints: each condition's own $R^2(\ell, t)$ (solid, seed mean; hairlines, seeds) against the un-anchored control's (dashed). Compare the weight profiles above: the pull's weight is far sharper than readability anywhere it concentrates.*

Redness is readable almost everywhere in the anchored runs. In every anchored condition, op1 reads at $R^2 \geq 0.90$ at every slice, so the anchor axis itself now encodes redness. Past the embedding, `+` and `=` read at 0.84 on average and never below 0.59, much as in the control: attention copies the operand's color into each syntax role, whether or not anything pulls there.

So the weight profile is not proportional to readability, and it couldn't be: at depth π̄ is one-hot, while R^2 is high at three roles of four. The softmin allocates by *alignment cost* among positions nearly all of which could host the concept, which is part of why the latch is a race rather than a search. The latched runs of `pool-t010` still read op1 at 0.92 at the last slice: the pull wasn't blind there; it had found `+` cheaper first.

The influence also runs the other way: where the pull keeps spending, it *creates* readability. op2 is the one role the control barely reads, at 0.19 across the two deepest slices. The span mean never stops pulling op2, and there it reads 0.71; the primary condition's pool withdrew from op2, and there it reads 0.22, back at the control's level.

Candidates we didn't run, queued in the project backlog:

- Per-color weight maps: do strongly-red lines concentrate differently from weakly-red ones?
- Where the `+` embedding drift comes from, and when: it persists even under the op1 oracle, as noted under H2. Recording drift per role in the trajectory would settle the timing.
- Two ways to avoid the latch: a τ schedule that starts soft and sharpens, or position dropout in the pool — mask a random subset of span positions out of the softmin each step, so no single position can absorb the whole pull early.
- More seeds at the chosen rung. Three per condition gives only a coarse estimate of the latch rate.

Discussion

What this experiment adds to the program is placement *along the sequence*. Where the concept lives in representation space is still chosen up front, as everywhere in SCA: the pull aims at the same fixed anchor direction throughout. The mellowmax adds one further choice: which token positions receive that pull. Left to choose, the pull chooses correctly at the embedding — the one slice where the architecture fixes the right answer, rather than the pull picking a favorite. Later milestones need that, because natural-language labels name concepts, not token positions.

The margin question mostly dissolved. Under the max-over-roles statistic, the span mean is already near the oracle, so position-finding was never going to improve the margin much. What it improves is the shape: the pooled pull withdraws drift from op2 and `=` nearly entirely, and the response it leaves at op1 grades better than the response under the span mean.

The qualification is the latch. Below $\tau = 0.1$, the deep-slice choice is a per-run race (see the readability probe above): it settles in the first few epochs, is never revisited, and a run that latches onto `+` loses some of the graded response that makes the anchor usable. So sharpening τ is not free — more concentration comes with more risk of committing to the wrong role before the representation has formed.

One possible remedy is a τ schedule that starts soft and sharpens as the geometry stabilizes. M1 makes that plausible: there, schedules that shaped the latent space at the right time worked where curricula didn't.[5] It may be hard to design in a larger model, though. A concept that only forms late in training would meet a pull that has already sharpened around whatever was cheap early. The schedule has to anticipate that timing, and three seeds per rung is too few to estimate the latch rates that would score it.

Mellowmax is an interesting finding for M3, where labels get coarser. Its softmin weights are the same per-position responsibilities that an EM-flavored label-assignment scheme would compute,[6] so the pooled term is already such a scheme, with single tokens as the unit. This experiment says it can locate a concept when the label covers a whole line, although our lines only had four tokens to pool over.

Coarser labels, whether per sentence, per document, or a sliding window with graded scores, would widen the pool: more candidates, and cheaper wrong ones. Nothing here rules that in or out, but this experiment does name the failure mode to design for: early winner-take-all latching, not diffuse smearing. The softest rung suggests the useful regime is soft pooling with a considered choice of *when* to sharpen, rather than hard selection from the start.

---

1. Ex-2.1.8 gated $\tilde{\alpha}$ at op1 only; the maxed-over-roles number is computed here from its published arrays. Note the repulsion already acts on every position — the syntax tokens climbed the axis despite it, because the span-mean pull itself pushes them there (three of the four pulled positions). Raising the repulsion floor would set up a tug-of-war on those same states, and ex-2.1.8 left that lever untested (its best floor was its highest). Pooling goes after the cause instead: it lets the pull withdraw from positions it doesn't need. ↩

2. The *residual stream* is the running vector each token carries through the network; a slice is that vector read off at one depth. ↩

3. Asadi & Littman, 2017. A temperature-controlled interpolation between min ($\tau \to 0$) and mean ($\tau \to \infty$); "softmin" is ambiguous between this and the softmax-weighted average, hence the care. ↩

4. The Spearman ρ grading track is retired as of this experiment. The ceiling analysis in ex-2.1.8 showed that ρ against `redness` can't reach its 0.8 gate alongside full-amplitude containment (best reachable 0.59–0.82), because the response target isn't monotone in redness. The r^2 track survives, but as a relative gate; ex-2.1.8 left the question of where a realistic absolute bar sits to a later grid. ↩

5. Regularizer weight schedules are an inheritance from M1. We haven't tested whether this architecture *needs* them: perhaps constant term weights would work. ↩

6. Expectation-maximization: alternate between assigning each item a soft share of responsibility to each component, and refitting the components to those shares. ↩