PROFORMA GLOBAL RESEARCH

Emergent Layer Roles and Functional Specialization

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-05-18

Executive summary. Inside a trained transformer, the question "what is each layer doing?" has a more concrete answer than the field usually allows. Using a small set of weight-forensic measurements (effective rank of the attention output projection, coefficient of variation across feed-forward neurons, saturated-position counts, and gradient-weight alignment) we classify every layer into one of four functional roles: **hub**, **damper**, **passthrough**, or **specialist**. The classification is reproducible from a finished checkpoint plus a brief gradient trace, and it predicts how the model responds to interventions. Two findings dominate. First, layer roles **migrate** under targeted regularization: when the mid-stack specialists in our baseline run were flattened (CV reduced 11×, saturated positions zeroed), specialist function did not disappear. It moved to the late stack, with effective rank held constant across the body. Second, **architecture shape controls where specialization concentrates**: a uniform-width variant pushes specialization aggressively into the final two layers (one layer reached CV=1.085 with 17% of its FFN positions saturated), while a variable-width variant at matched parameters distributes the same specialist function across five late layers. The deeper observation behind both findings is the same: uniformity is the default attractor of trained transformers, and specialization requires asymmetric pressure. The functional organization of a trained model is not an immutable property of the architecture: it is a controllable property of the training process.

1. Why this paper exists in the series

The three preceding papers describe a deliberate progression. Paper 1 set the methodology: every hand-picked number is a temporary scaffold around a missing reward path. Paper 2 documented the foundational result: a REINFORCE Sidecar operating on the forward-pass control surface beat a hand-tuned vanilla baseline by 42% at matched compute, and additional control channels added incremental gains on top. Paper 3 extended the same paradigm one level up the stack, to the model's own architecture, and documented both the substrate that lets a policy mutate per-layer widths mid-training and the optimizer-state coupling that the foundational architectural-mutation run had to confront.

Each rung in that progression changes the body of the trained model. The vanilla baseline trains one kind of body; the Sidecar with the foundational channel set trains a different one; the Sidecar with the added per-tensor weight-decay channel trains a third; the architectural-mutation substrate trains a fourth. Comparing eval metrics across these runs tells us *which* rung

wins. It does not tell us *what each rung did inside the body*: how the layers reorganized, where computation concentrated, what kinds of internal structure each intervention produced or suppressed.

This paper develops the forensic framework that answers that question. The methodology is general (it applies to any decoder-only transformer) and cheap (it requires nothing beyond a finished checkpoint and a brief gradient trace), and it is the diagnostic we use to read what each step in the progression has done. Two specific case studies in the back half of the paper compare body organization across rungs (vanilla vs. Sidecar-regularized, uniform-width vs. variable-width) and demonstrate the kind of structural finding the framework surfaces.

2. The question

Mechanistic interpretability has produced a rich vocabulary for describing what individual circuits inside trained transformers do: induction heads, copy-suppression circuits, anti-induction heads, indirect-object identification. Most of that vocabulary is built bottom-up, from the discovery of specific computations inside specific small models. It is also expensive: tracing a circuit requires hand-built probes, careful activation patching, and a willingness to commit weeks to one model.

For a research program that needs to evaluate dozens of training runs per month, that approach is unaffordable. We need a coarser, cheaper, top-down question: **what kind of layer is layer *i* in this checkpoint?** Not "what specific circuit lives there": *what role does it play in the body's overall computation?* The answer must be computable from a finished checkpoint and a brief gradient trace, must be stable enough across training conditions to compare runs, and must change meaningfully when something about the run changes. That is the taxonomy we build in this paper, and the use we put it to.

The headline observation we develop below is that **layer roles are not invariant features of the architecture; they are emergent properties of the training dynamics**, and they can be moved around with targeted pressure. The body of a trained transformer behaves as though it has a roughly conserved budget of "specialist function" (concentrated, high-variance computation that lives in a small number of FFN positions and attention head-pairs) and where that budget lives is controllable. Flatten one region with regularization and another region absorbs it. Change the architecture's width profile and the budget migrates toward the end of the stack or distributes across more layers.

3. A taxonomy of layer roles

We classify each layer into one of four roles based on a small number of weight-only statistics. The labels are deliberately coarse (we are not claiming that every layer falls cleanly into exactly one role, and Section 4 explains the tie-breaking), but the four-way split captures the qualitative differences we observe across hundreds of trained variants.

Hub. A hub layer routes information widely. Diagnostics: high effective rank in the attention output projection W_o (the layer mixes many directions, not a few); balanced Frobenius energy between the attention block and the FFN block; broad weight magnitude distribution across heads, with no single head dominating; high coupling to multiple other layers in cross-layer head-fingerprint comparisons. Hubs are the aggregation points of the body: when a downstream layer needs information from many earlier positions, it most often pulls it from a hub.

Damper. A damper layer suppresses residual stream magnitude. Diagnostics: large Frobenius norm in the FFN down-projection (relative to the layer's own gate/up norms); negative or near-zero cosine alignment between Adam's momentum vector and the current weight in the down-projection (the layer is being pushed to *shrink* the residual it writes); high "stuck" fraction in the optimizer state (parameters whose first- and second-moment estimates are both small, i.e., dead regions of the tensor). Functionally, dampers act as runtime regularizers: they prevent the residual stream from exploding as the body composes more contributions.

Passthrough. A passthrough layer contributes a near-identity transformation to the residual stream. Diagnostics: low CV in FFN co-magnitude (defined below), no saturated FFN positions, low "total write" magnitude relative to peer layers, and high adjacent-layer activation alignment. Most early and mid layers in most trained models we have built are passthroughs. They are not idle, but their contribution is broad, smooth, and small.

Specialist. A specialist layer memorizes or implements specific patterns. Diagnostics: high CV in the FFN co-magnitude profile, multiple FFN positions whose gate×up product exceeds 3σ above the layer mean (we call these **saturated positions**), concentrated attention on specific head-pairs (high head-diversity score, multiple unique-role head clusters). A specialist is the model's lookup table: when an input matches one of the patterns it has memorized, the specialist fires hard; on most inputs, it sits quiet.

4. Methodology: how the classification is computed

The classification is built from four primitive measurements per layer, plus a few cross-layer ones. Each is cheap to compute from a finished checkpoint and a short gradient trace over the final training epoch.

Effective rank of W_o. We compute the singular values of the attention output projection, normalize them to a probability distribution, and take the exponential of the Shannon entropy: `effective_rank = exp(H(σ))`. This gives a continuous measure of how many "effective dimensions" the layer mixes. A layer that uses every direction equally has effective rank equal to the matrix's full dimension; a layer that routes through a few dominant directions has much lower effective rank. We use the same primitive on the FFN gate and down matrices.

FFN co-magnitude profile. Each FFN intermediate position *p* corresponds to one row of W_gate and one row of W_up. We compute, for every *p*, the product `|W_gate[p, :]| · |W_up[p, :]|`. This product measures how active position *p* is in the layer's computation: high values mean both the gate and the up-projection direct large energy to that neuron. We summarize the per-layer profile by the coefficient of variation (std / mean across the intermediate dimension), and by the count of "saturated" positions whose product exceeds three standard deviations above the layer mean. A layer with low CV and zero saturated positions is using its FFN diffusely; a layer with high CV and many saturated positions has carved out a small number of high-energy neurons for memorization.

Attention/FFN balance. The Frobenius energy ratio between a layer's four attention tensors (W_q, W_k, W_v, W_o) and its three FFN tensors (W_gate, W_up, W_down) tells us whether the layer leans on attention mixing or feed-forward computation. Hubs sit near 0.4 attention share; specialists and dampers can lean either way; pure FFN-end layers can drop attention share below 0.3.

Cosine alignment of momentum with weight. Over the last *k* steps of training we record Adam's first-moment estimate per tensor and compute the cosine between that vector and the current weight. Positive alignment means the optimizer is still pushing the layer in the direction it has been growing: the layer is still being shaped. Negative alignment means the optimizer is trying to unwind something. Near-zero alignment means the layer has settled.

The four primitive measurements feed a small number of derived features: head_diversity, unique_role count from clustering FFN positions, gamma concentration in the post-norm scale, copy-density and total-write magnitudes from a brief activation probe. Each feature is z-scored across the layer dimension and routed to the role it is evidence for. A sigmoid of the averaged z-scores becomes the per-role score; the layer's primary role is whichever score wins

by margin > 0.05, otherwise it is recorded as a tie. The full feature list and the routing matrix are implementation detail; the principle is that the role classification is a *function of the checkpoint*, not a result of additional training or hand-built probes.

A representative classification from one of our long-trained 24–layer variants at 102K steps:

```
L0 specialist (head_diversity, ffn_cluster_entropy dominant)
L1 hub (high coupling, sink_mean, synapse_mediator)
L2 damper (high post-norm damped fraction, low eff_rank wdown)
L3 passthrough
L4 specialist (mild)
L5 passthrough
L6 passthrough
L7 damper
L8 damper
L9 hub
L10 passthrough
L11 hub
L12 hub (cross-layer head-match dominant, the "spine")...
L20 hub
L21 specialist
L22 passthrough
L23 damper (low source_mean, low eff_rank wdown, final pre-head)
```

Three points are worth pulling out. First, the very first and last layers are *not* passthroughs: they are specialist (L0) and damper (L23), reflecting the model's need to project tokens into and out of the residual stream. Second, there is a clear "spine" of hubs at depth (L1, L9, L11, L12, L16, L20) that the body uses as routing anchors. Third, dampers cluster at specific depths (L2, L7, L8, L18, L23) where they keep residual magnitude under control between bursts of specialist computation. None of this was designed; the body discovered it.

5. Headline finding 1: layer roles migrate under targeted regularization

Our reward-driven training control work (see *"Reward-Driven Training Control"*) introduced per-(layer, tensor) learned weight decay: a small policy that proposes a weight-decay coefficient for each of the body's tensors, with the proposal updated by REINFORCE against held-out loss. Two runs let us isolate the structural effect of that control: a **baseline run** with uniform

global weight decay (Run A) and a **regularized run** with per-tensor learned weight decay applied (Run B). Architectures were identical (11 layers, ~478M parameters); the only difference was the regularizer.

Run B improved held-out perplexity. The structural question, beyond the headline number, is what changed inside the body.

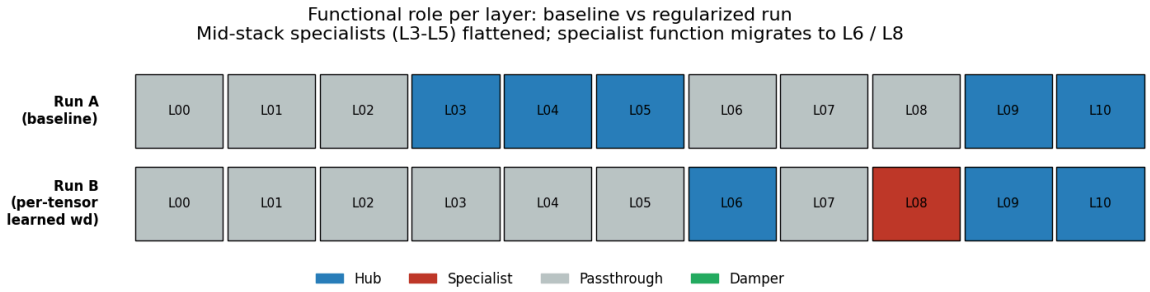

Figure 1. Functional roles per layer in the two runs. Mid-stack specialists L3–L5 in Run A flatten to passthrough in Run B; specialist function migrates to L6 and L8.

The mid-stack region (L3, L4, L5) was the most concentrated, specialized region in Run A. L5 had CV = 0.732, the highest in the model, with three saturated FFN positions and an extreme outlier weight in its gate projection (max-abs = 0.55, two to ten times peer layers). L4 had CV = 0.270 with six saturated positions and the largest single-tensor change across the entire training run. L3 had CV = 0.264 with five saturated positions, plus the largest attention-side outlier in its key projection.

Under the per-tensor regularizer, those three layers were demolished as specialists:

layer	CV (Run A)	CV (Run B)	flattening	saturated positions A→B
L3	0.264	0.075	3.5×	5 → 0
L4	0.270	0.041	6.6×	6 → 0
L5	0.732	0.064	**11.4×**	3 → 0

L5's coefficient of variation dropped by more than an order of magnitude. L4's outlier weight collapsed from 0.35 to 0.06 (84% peak reduction). The saturated FFN positions in all three layers went to zero.

This could have meant the body lost capacity: that the regularizer simply suppressed an entire region of computation. It did not. Effective rank of W_o, measured on every layer in both runs, stayed within ±3 across the entire body. The FFN matrices behaved the same. The mid-stack flattening was not a loss of capacity; it was a redistribution.

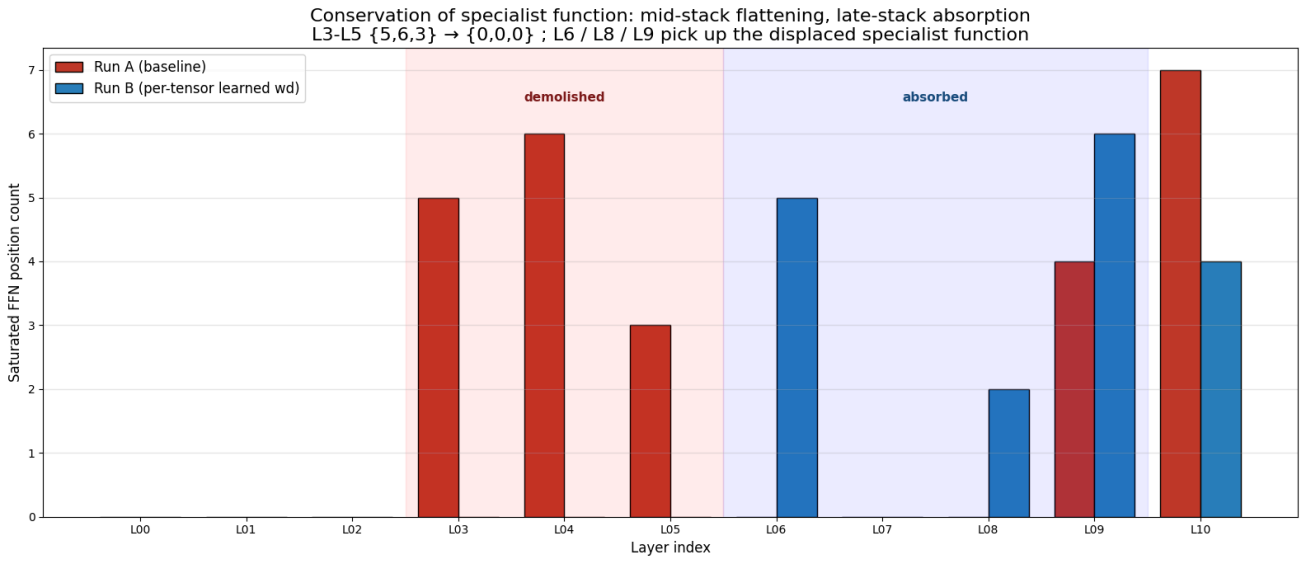

Figure 2. Saturated FFN position counts per layer. The five-six-three positions in L3–L5 of Run A move into L6, L8, and L9 of Run B. The total across the body is approximately preserved.

The displaced specialist function emerged downstream. L6 went from a passthrough layer in Run A (CV = 0.069, zero saturated positions) to a new hub in Run B (CV = 0.296, five saturated positions). L8 went from passthrough to mild specialist (CV = 0.080 → 0.125, two saturated positions). L9, already a hub in Run A, deepened (saturated count grew from four to six). The total saturated count across the body's middle and late stack was approximately preserved.

We interpret this as conservation. The body behaves as though it has a roughly fixed budget of "specialist function" to allocate across the depth dimension. The budget is what the training data demands: it is set by the amount of pattern-specific memorization the loss surface rewards. The *location* of the budget is what the training dynamics decide, and the training dynamics are something we can shape. Flatten one region and another absorbs it. The model is not destroying computation; it is moving it.

The practical implication: the diagnostic that detects pathological specialization (extreme CV, extreme outlier weights in a single mid-stack layer) is also the diagnostic that detects when the regularizer has succeeded in relocating that specialization to a more useful position. Late-stack specialization, closer to the output projection, appears to be more useful than mid-

stack: both because Run B improved on Run A's eval, and because the structural pattern (concentrated computation in the layers right before the language-model head) matches the kind of work the model has to do in the last layers regardless.

6. Headline finding 2: architecture shape controls where specialization concentrates

The conservation observation invites a follow-up question: if the body redistributes specialist function freely, does the architecture's shape control where it ends up *by default*, before any regularizer is applied?

To answer that we compare two runs that share everything except width profile. Both are 11 layers, both around 478M parameters, both trained with the same reward-driven regularization. The **variable-width variant** uses a per-layer hidden ramp (hidden dimensions 1024 in the early stack growing to 1372 in the late stack). The **uniform-width variant** uses 1168 hidden dimensions at every layer: same parameter budget, no asymmetry.

Figure 3. Coefficient of variation of FFN gate×up co-magnitude, per layer, in the two architectural variants. Early and mid layers are diffuse in both. Late-stack specialization concentrates dramatically in the uniform variant: L10 reaches CV=1.085 versus 0.276 in the variable variant.

Through L0–L5 the two runs are indistinguishable. Both have diffuse, low-CV early and mid stacks. This is itself informative: the regularizer (which fires on both runs) demolishes mid-stack specialization regardless of width profile. The architectural difference does not show up until L6.

From L6 onward, the runs diverge. The variable-width variant distributes specialist function across five layers (L6 through L10), with L6 acting as the first late-stack hub (CV = 0.296) and L10 as a moderate hub (CV = 0.276). The uniform-width variant compresses the same function into the last two layers. L10 in the uniform variant reaches CV = 1.085, nearly four times the variable variant's L10, and 609 of its 3504 FFN positions (17.4%) cross the saturated threshold. L9 in the uniform variant reaches CV = 0.512 with 114 saturated positions. The mid-late layers (L6, L7, L8) in the uniform variant are quieter than in the variable variant; the specialist budget has slid down toward the head.

Figure 4. Effective rank of the attention output projection per layer. The uniform-width variant carries +30 to +46 effective dimensions everywhere: equal width gives every layer the same representational headroom.

The effective-rank picture is the other half of the story. The uniform variant carries +30 to +46 more effective dimensions in W_o at every depth. Equal width gives every layer the same representational headroom; the variable variant front-loads narrower widths into the early stack to save parameters for the late-stack ramp, and the narrow early layers carry less rank. The trade is real: the uniform variant has more headroom per layer, the variable variant has different shapes per layer. The variable variant pays for those shapes with rank.

At held-out evaluation the uniform variant outperforms the variable variant, modestly but consistently, around 5% relative on our primary metric. The interpretation we draw from the structural analysis: **the late-stack concentration is doing useful work**. Concentrated specialist computation in the layers immediately before the projection back to the vocabulary is, on this task mix, more valuable than distributed specialist computation across the late half of the body. The variable-width variant dilutes the specialist budget across more layers and, in doing so, dulls the model's ability to perform sharp final-stage routing.

There is a corollary worth surfacing. The L10 in the uniform variant has an attention proportion of 28.2%, 11 percentage points lower than the variable variant's L10. Combined with the extreme FFN concentration, this means **L10 in the uniform variant is essentially a pure FFN layer**: minimal attention mixing, near-maximal FFN specialization, sitting right against the output. The model chose this shape, neither the architecture nor the regularizer prescribed it, and it chose differently when the architecture's width profile gave it room to spread the load.

7. Headline finding 3: uniformity is the default attractor

Both of the previous findings rest on the same underlying observation, which has become the central design principle of our entire research program: **uniformity is the default attractor of trained transformers**.

When every layer has the same hidden dimension, the same head count, the same FFN width, the same learning rate, the same gradient-clip threshold, the same activation function, the gradient signal at every layer is symmetric with respect to depth. The loss surface has no preference between configurations in which layer *i* and layer *j* play swapped roles. The optimizer rotates the weights into a subspace where every layer looks like every other layer, because there is no reward for picking one over another.

We have seen this in our long-trained baselines. A vanilla 24-layer transformer trained on our task mix lands with an attention/FFN-balance distribution that spans only 0.43 to 0.48: every layer indistinguishable on the write-ratio measurement, standard deviation about 0.014. Compare a trained variant from our reward-driven lineage, which spans 0.43 to 0.84 with two FFN super-hubs and a band of FFN-dead mid-layers in between: the standard deviation is about 0.13, ten times the baseline.

The reward-driven variant did not start more asymmetric. It started identical. What changed is that the reward path exposed an asymmetry (per-tensor weight decay, per-layer learning rate scale, per-(layer, axis) architectural mutation) and the model's own credit assignment broke the symmetry. Asymmetric pressure is the necessary condition for specialization. Without it, depth is wasted.

The corollary is just as important. Every symmetry in the architecture is a *latent differentiation axis* waiting for a reward path. Same head dimension across heads is an axis. Same attention/FFN ratio across layers is an axis. Same hidden dimension across layers is an axis. Same residual weighting is an axis. Same RMSnorm scale initialization is an axis. Each of these can be unlocked by exposing the parameter to a reward signal: at which point the model will use it to break the symmetry in whatever direction the data rewards. (Each unlocked axis carries a footgun risk: an early version of per-layer learning-rate scaling, with an unsafe minimum-epsilon clamp on Adam, destroyed training. The fix is to sanity-check clamp ranges against the downstream arithmetic before wiring a new axis, but the broader principle stands.)

This is the unified statement of the project's vision. *Turn data into structure.* The data has intrinsic structure: token distributions, circuit demands, problem topology. Uniform architectures cannot match that structure because they are symmetric by construction. The path from uniform to structure-matching runs through asymmetric controllable parameters wired to reward signals. Every paper in this series is an instance of this principle. Reward-driven training control breaks the symmetry of optimization. Self-discovering architectures break the symmetry of width. Layer-role classification, the subject of this paper, is the post-hoc forensic that lets us see how cleanly the symmetry got broken, and where it didn't.

8. Diagnostic uses of the taxonomy

A layer-role classifier that takes minutes to run on a finished checkpoint is useful because it answers questions that previously required hours of activation patching or weeks of mechanistic probing. We have built three operational uses for it.

Long-trajectory analysis. We track layer roles every 5K-10K steps over the full course of a 100K-step training run. Two patterns emerge consistently. First, roles **stabilize early**: by step 20K-30K most layers have committed to their eventual role, and after that the role distribution evolves slowly. Second, the discontinuous loss drops we observe late in training (we have seen Math perplexity drop from 706 to 434 in two consecutive thousand-step intervals) **coincide with role transitions**. Typically what happens is that a passthrough layer transitions to a specialist as a new pattern locks in, or a hub deepens its routing on a previously underweighted head-pair. The classifier gives us a way to see these transitions without manually inspecting weights.

Detecting training pathology. Pathological runs show a characteristic signature: late-stack FFN body weights with spectral norms 30 to 100 times their peer layers. The specific instance we first encountered was in an early variable-width experiment, where the layers added at the late stack (L9 and L10) developed gate-projection spectral norms around 87 and 141 while peer layers sat between 5 and 12. In a transformer without a router around the affected layers, this means FFN output per token grows to magnitudes that overflow the residual stream, and training NaN's. The pathology is invisible to the loss curve until it hits: by step 3K the layer norms are already 10–20× peer; the NaN comes at step 4400. The classifier surfaces this kind of pathology at every regular check, well before the run dies, and we use it to decide whether to introduce mid-stack pressure (per-layer LR scaling, per-tensor weight decay) on a specific layer before the explosion turns into a crash.

Inference-time interpretation. A model whose layers are classified gives us a coarse map of which depths to ablate when investigating capability, which to extend when scaling, and where to insert new structural features. Conditional-routing modules go on the downstream side of hubs (they are the aggregation points; routing from a hub lets one module see everything summarized). Pattern-finding modules branch at specialist depths (the model already has circuit-specific computation living there). Passthrough layers are the natural places to insert skip-gates (the model already behaves as though they add little). We have used this to guide several rounds of architectural experimentation; the experiments that respected the classification consistently outperformed those that placed new modules in arbitrary positions.

9. An unexpected finding: register tokens worked as random-init attention sinks

One experimental variant from this lineage added a small set of additional non-token positions to the input sequence ("scratchpad" tokens whose role was to absorb attention from real tokens and provide a place for transient computation). The design hypothesis was that the

model would learn meaningful content into these positions: their embeddings would shift over training as the model discovered what to write there.

It didn't. Across 15,000 training steps the relevant embedding RMS moved from 0.01807 to 0.01812, a change of 5×10^{-4}, roughly 30× smaller than the change a directionally-consistent gradient at that learning rate would have produced. The gradient was flowing in (we verified by capturing pre-AdamW telemetry on a follow-up run); it was just noise in direction, not signal.

And yet the mechanism worked. Held-out perplexity improved by 11–16% mid-run versus the baseline without scratchpad tokens. The tokens served their function, but not as learned scratch space. They served as Darcet-2024–style **attention sinks**: fixed K/V slots that real tokens could divert attention into, randomly initialized and effectively unchanging across training, but providing the structural slack the attention distribution needed to avoid over-concentrating on real-token positions.

This is worth publishing precisely because it isn't the result we predicted. The design was wrong about the mechanism. The outcome, improved perplexity, was right. Negative-result-with-positive-takeaway findings are exactly the kind of evidence that distinguishes a research program from a marketing pitch, and we have made it a working norm to document them rather than retroactively rationalize them.

There is a deeper diagnostic point here, too. The reason we identified the mechanism difference is the same forensics methodology this paper develops. We measured the embedding's RMS trajectory over the run, compared it to the expected magnitude under the actual learning rate, and noticed the 30× gap. Without the per-tensor weight-state forensics that underpin the role classification, the finding would have read as "scratchpad tokens help. We don't know why" rather than "scratchpad tokens help; the embeddings don't learn; the mechanism is random-init attention sinks."

10. Why this work matters externally

The interpretability literature on transformers tends to treat a trained model as a fixed artifact and ask what is inside it. This paper takes the inverse stance: the functional organization of a trained transformer is **a controllable property of the training process**, and the REINFORCE Sidecar (see *"Reward-Driven Training Control"*) and its architectural-decision extension (see *"Self-Discovering Architectures"*) together give us the policies that control it. The role classification developed here is the diagnostic that lets us read what those policies did to the body. This forensic capability is what closes the loop on the unified RL self-tuning thesis: a Sidecar that delivers a 42% over-vanilla improvement is meaningful primarily as a number, but a

Sidecar whose effect on the body's functional organization can be *read* (whose interventions can be diagnosed at the level of which layers shifted role, where specialization migrated, what redundant capacity was removed) is meaningful as a *paradigm*, because the diagnostic is what lets the next iteration of the policy improve on the current one.

This matters externally in three ways.

First, it changes the unit of analysis. Capability claims about transformer-based systems usually pin to whole-model evaluations (perplexity, downstream task accuracy, generation quality). Those are necessary but not sufficient. A model can match another model's eval while having an internal structure that is more or less suited to extension, ablation, or fine-tuning. Two of our runs that matched on aggregate eval differed sharply in role distribution (one concentrated specialist function in two late layers, the other distributed it across five) and these differences predict how each model responds to subsequent intervention. Eval is the boundary condition; structure is the explanation.

Second, it lets us **shape, not just observe**. The combination of reward-driven training control, self-discovering architectures, and layer-role classification means we can design experiments that say "we want the body to organize like this" (late-stack specialization, mid-stack damping, hubs at specific depths) and then check whether the training run delivered that organization. When it doesn't, we know what to change in the next iteration. This is a tighter feedback loop than the field's standard "train, eval, retrain larger."

Third, it scales. The methodology (effective rank, FFN co-magnitude, saturated-position counts, gradient-alignment) is generic. It applies to any decoder-only transformer (and most of it generalizes to encoders) without modification. We have run it on body sizes from 11 to 24 layers and on parameter counts from 200M to 500M. It does not require activation tracing, does not require labeled probe data, and does not depend on the specifics of the task mix. A practitioner who wants to compare two trained models on structure rather than on aggregate eval can adopt this framework with no infrastructure beyond a checkpoint loader.

The methodology is publishable. What we hold back is the implementation detail of the levers (the reward formulas, the policy parameterizations, the credit-assignment paths) and of the asymmetric pressures that make the role distribution shapeable. Those are what turn the diagnostic from a measurement into a control system. The combination of the diagnostic and the control system is the firm's depth.

This is Paper 4 of a four-paper series. Paper 1 ("Research Posture: Why We Let the Model Choose") develops the iteration methodology that produced these results. Paper 2 ("Reward-Driven Training Control") introduces the REINFORCE Sidecar paradigm and documents the foundational +42%

over-vanilla result whose downstream effects on the body this paper measures. Paper 3 ("Self-Discovering Architectures") extends the Sidecar paradigm to architectural decisions and documents the substrate whose chosen shapes are also subjects of the forensic framework developed here.
