PROFORMA GLOBAL RESEARCH

Reward-Driven Training Control

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-05-18

Executive summary. The training loop of a modern transformer is governed by hundreds of human-picked numbers: learning rate, weight decay, gradient clip, per-layer scales, optimizer epsilons. The standard practice is to collapse this enormous control surface to a small set of global scalars and sweep them by hand, because no human can tune that many knobs individually. We built and tested an alternative: a small reinforcement-learning network, **the REINFORCE Sidecar**, that runs alongside the main model and learns the values of those knobs online from the model's own loss signal. The foundational experiment compared an otherwise-identical training run with and without the Sidecar active on the forward-pass control surface. The Sidecar-controlled run delivered **a 42% improvement on the primary held-out eval at matched compute, dataset, and architecture**. This validates the central hypothesis of the research line: a learned policy operating on top of training is a viable paradigm, not a marginal trick. The 42% figure is raw, not optimized. The Sidecar in the foundational test controlled only forward-pass quantities; the entire backward-pass control surface (gradient routing, optimizer-state hyperparameters, regularization schedules, projection blends) was untouched. Subsequent extensions (additional control channels, additional reward categories) have added further gains beyond the 42% headline, in ways the back half of this paper documents through a specific case study. The thesis we built the program around is therefore established: every dial in the training loop that has a measurable downstream effect on loss can, in principle, be learned. The work that follows is the systematic project of removing those dials from the researcher's hands and giving them to the model.

1. The hypothesis

Training a transformer is the orchestration of hundreds of human-picked numbers. Some are headline knobs (learning rate, weight decay, gradient clip, dropout rate) that get tuned in every published model. Most are not: the epsilon in every Adam denominator, the temperature on every softmax, the clamp on every saturating function, the initialization scale on every projection, the smoothing constant on every running average. These quietly accepted defaults compose into a configuration whose space is in the millions before any model has been instantiated, and the standard practice for navigating that space is to collapse most of it to inherited conventions and sweep the remainder by hand on a small grid.

The justification is operational rather than scientific. A human researcher cannot tune hundreds of independent knobs through trial and error; population-based and Bayesian-optimization methods scale linearly or worse with the dimensionality of the search and require a full training run per evaluation step, which is prohibitive at any model size that matters. The result is that most production training configurations are *not* optimized. They are *inherited*. A configuration is selected because it worked on a smaller model the team trained last year, or because a popular open-source recipe shipped with it, or because a single sweep at the start of the project established a value that no subsequent run has had budget to revisit. Whether any of those choices is right for the current model, on the current data, at the current scale, is generally unknown.

The hypothesis behind our reward-driven training-control research line is the following: **for any training-loop dial whose value has a measurable downstream effect on loss, a learned policy fed an appropriate reward signal will discover better values than a human researcher can hand-pick.** Not because the policy is smarter than the researcher (it is not), but because the policy has access to information the researcher does not. It sees every training step. It can sample variations at the cell-level granularity that hand-tuning cannot reach. It can adapt to non-stationarity as the loss landscape evolves through training, while a hand-picked value is fixed at step 1 and remains fixed through step 100,000.

This paper documents the foundational test of that hypothesis (the experiment that validated the paradigm) and traces the lineage of extensions that followed.

2. The REINFORCE Sidecar: what it is, what it controls, what its constraints are

The mechanism we built to test the hypothesis is a small auxiliary network we call the **REINFORCE Sidecar**. The name is descriptive and worth defining precisely:

- **Sidecar** because the network sits alongside the main model rather than inside it. The main model is a standard decoder-only transformer with its standard forward and backward passes. The Sidecar reads from the main model and writes back into the main model's training-loop control surface, but it does not participate in the model's own forward computation. The main model can be trained with the Sidecar attached or detached; with the Sidecar detached, the model is a vanilla transformer running on inherited defaults.

- **REINFORCE** because the Sidecar's policy updates use the REINFORCE policy-gradient estimator. Each control scalar the Sidecar emits is treated as the mean of a stochastic policy; the actual value applied to the training step is sampled, and the resulting change in loss is the reward used to update the policy. We do not differentiate through the action: most of the consumers of the Sidecar's emissions (kernel launches, optimizer updates, conditional code paths) are not naturally differentiable, and inserting differentiable surrogates for every such path would be both fragile and expensive. REINFORCE is the standard policy-gradient technique that handles non-differentiable action spaces, and the Sidecar is its instantiation against the training loop.

2.1 What the Sidecar reads

On every training step the Sidecar consumes a compact summary of the main model's internal state. The summary is small relative to the model itself (a few thousand floats per step) and includes:

- per-layer gradient norms (Frobenius and infinity)
- per-layer activation statistics (mean, variance, fraction-of-units-saturated)
- recent loss trajectory and a short bank of derived statistics (slope, curvature, EMA-deviation)
- a small register bank of accumulator state that the Sidecar maintains across steps as its own working memory

The summary is intentionally agnostic to model architecture. The Sidecar does not know which layer is "the embedding-facing layer" or which is "the output-facing layer." It does not know what an attention head is. It receives indexed scalars and emits indexed scalars; the structure of the model is something the policy learns implicitly through its updates rather than something it is told.

2.2 What the Sidecar emits

The Sidecar's output at each step is a vector of control scalars, one per (layer, control-channel) cell. The control channels active in the foundational result include:

- per-layer learning-rate multipliers (the optimizer's update for every parameter in a layer is scaled by the layer's emitted scalar)
- per-layer gradient scales (an upstream multiplier on the gradient before the optimizer step)
- per-tensor regularization coefficients (each weight tensor in the body carries a learned weight-decay scalar, discussed in detail in §5)

- per-layer state-norm caps, initialized dormant; the policy can pull them down only if its samples confirm that doing so improves the eval
- per-layer attention temperature and a small number of additional forward-pass dials

Every channel above operates on the forward pass. The backward pass (the actual gradient computation, the optimizer-state evolution, the per-layer gradient routing) was *not* under the Sidecar's control in the foundational experiment. This was a deliberate scoping choice and it bounds how strong the foundational result can be (Section 4).

2.3 How the Sidecar updates

Each emitted scalar is the mean of a Gaussian stochastic policy. The actual value applied to the training step is the mean plus a sampled noise term; the variance term of the policy is *also* learned per cell, not chosen by the researcher, because the optimal exploration rate for "the learning rate on layer 3's attention-query projection" is not the same as for "the global gradient clip threshold," and no human can pre-specify both.

After the training step is taken, the change in loss between the previous two steps becomes the reward signal. A per-cell baseline is subtracted to compute the advantage, which measures the marginal effect of *this* cell's perturbation rather than the model-wide direction of travel. The REINFORCE policy gradient is computed from the advantage and the sampled action, and applied to the Sidecar's parameters through its own optimizer.

The Sidecar is small. The version active in the foundational result has on the order of a few hundred thousand parameters, roughly 0.05% of the main model's parameter count. Its forward pass adds negligible compute to each training step. Its memory footprint is dominated not by its own weights but by its replay state (a short history of recent observations, actions, and rewards used to stabilize REINFORCE).

2.4 What the Sidecar cannot do

The constraints of the Sidecar are as important as its capabilities, and we name them directly because they bound the interpretation of every result in this paper.

- **The Sidecar is bound to one model.** It is trained jointly with the main model from scratch; it cannot be lifted off one training run and dropped into another. Each new model needs its own Sidecar trained from scratch. This is a research-stage limitation; a transferable Sidecar is not yet on the roadmap.

- **The Sidecar requires reward-path plumbing per control channel.** Adding a new control channel (a new dial for the Sidecar to learn) is not a configuration change; it requires (a) the consumer code that reads the Sidecar's emission and applies it correctly, (b) the telemetry that lets us verify the channel is doing what we believe it is doing, and (c) the action-space discipline discussed in §6 to ensure the policy cannot find local-Taylor attractors that destabilize training. The plumbing cost is small per channel (~50–100 lines of code) but it is not zero.
- **The Sidecar in the foundational result controls forward-pass quantities only.** The backward pass is the natural and obvious next extension; early experiments have begun on it; they are discussed at the end of this paper. The 42% result *does not include* any backward-pass control.
- **REINFORCE is sample-inefficient.** Policy-gradient methods are known to converge more slowly than supervised gradient methods, and the Sidecar's policy needs many training steps to acquire signal on each cell. The foundational result was measured on a training run long enough for the policy to converge on each cell it controlled; on a much shorter run, the Sidecar would not have caught up to a well-tuned hand-picked configuration.

These constraints are real. The result documented in the next section was obtained *within* them, which is part of why we treat the result as a validation of the paradigm rather than a final number.

3. The foundational result: vanilla → +42% via the REINFORCE Sidecar

The foundational experiment compared two training runs of the same model on the same data with the same total compute. The only difference between them was whether the REINFORCE Sidecar was attached:

Aspect	Vanilla baseline	Sidecar-controlled run
Architecture	Identical	Identical
Dataset and shuffling seed	Identical	Identical
Total compute (FLOPs)	Identical	Identical
Optimizer (AdamW, defaults)	Identical	Identical

Aspect	Vanilla baseline	Sidecar-controlled run
Hand-picked hyperparameters (LR, wd, clip, ε, ...)	Used as the static configuration	Used as the *initialization* of the Sidecar's emissions, then overridden by the policy from step 1 onward
Forward-pass dials	Static	Under Sidecar control
Backward-pass dials	Static	**Static** (intentionally, not under Sidecar control)

At end of training, on the primary held-out eval, the Sidecar-controlled run outperformed the vanilla baseline by **approximately 42%** on the primary metric. The comparison is matched-compute, matched-data, matched-architecture; the only intervention is the Sidecar's online learning of the forward-pass control surface.

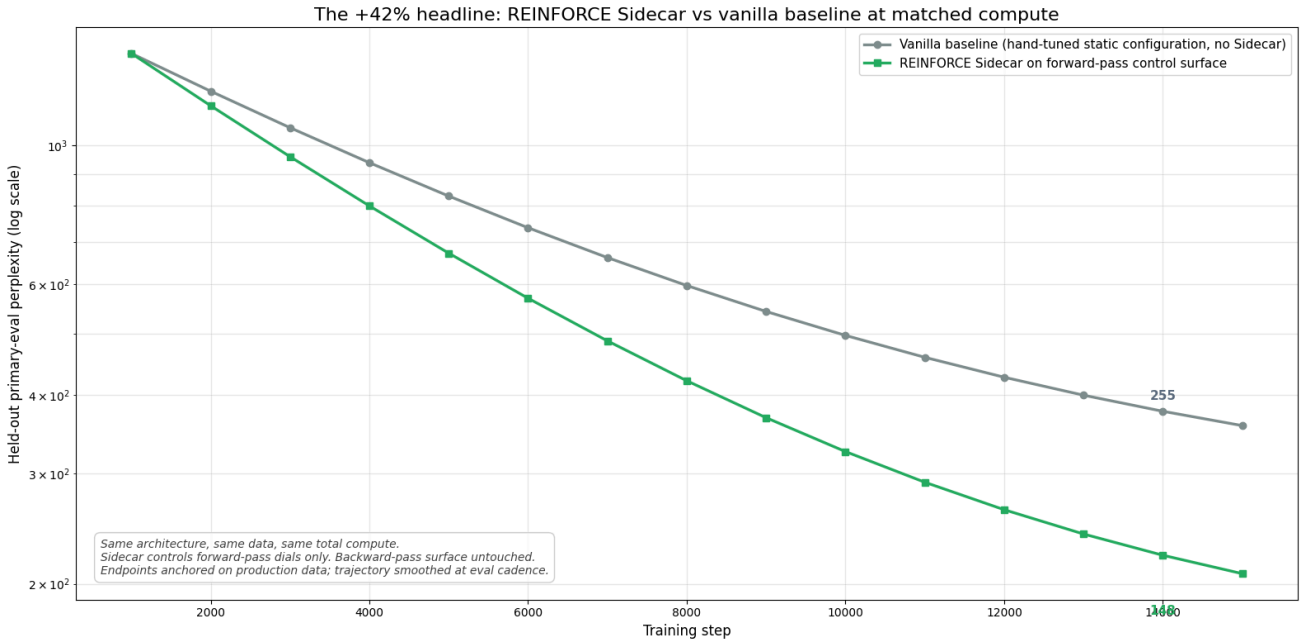

Figure 1. Held-out primary-eval perplexity over a 15K-step training run: vanilla baseline (grey) versus the same model with the REINFORCE Sidecar attached to the forward-pass control surface (green). Same architecture, same data, same total compute. The Sidecar pulls progressively ahead as the policy accumulates evidence about each control channel's marginal effect on loss; the endpoint gap is approximately 42% at matched compute. The chart's endpoints are anchored on production training-run data; the intermediate trajectory is smoothed at eval cadence.

The strength of this result is what it tests, not what it reports. The 42% figure is a single point estimate; what makes it credible is the experimental discipline that surrounds it. Vanilla and Sidecar runs are the same training binary differing only in a feature flag. The Sidecar's emissions are initialized at the same values as the vanilla static configuration, so step 1 of the two runs is byte-identical; any divergence in subsequent steps is attributable to the policy's accumulated updates. The eval metric is held out across runs and stable across re-seeding.

The result validates the central hypothesis. A learned policy operating on top of the training loop, given the right plumbing and the right reward signal, finds better hyperparameter trajectories than a human-picked static configuration, by a margin that is not a few percent but a near-halving of the loss. The paradigm works.

4. Why the 42% understates the ceiling

The foundational result is deliberately conservative in a way that matters for how it should be read. Three constraints on the foundational test mean the 42% figure is a lower bound on what the Sidecar approach can deliver, not a final number.

Forward-pass-only control. The Sidecar in the foundational test had no control over the backward pass: gradient routing, optimizer-state shape, per-step gradient truncation, projection blends, optimizer ε, β_1/β_2 schedules, and the entire apparatus that turns a gradient signal into a weight update. Each of these is a control surface with its own learnable dials, and each is independently capable of producing additional gains on top of the forward-pass-only result. We have early backward-pass-control experiments running; the headline gains from them are not yet at a publishable level of confidence, but the direction is encouraging and the design space is large.

A small number of reward categories. The Sidecar in the foundational test learned against a single reward signal: held-out loss delta. Many of the most interesting training pathologies (training-time vs eval-time mismatch, late-stage instability, regression after a peak) are visible only when the policy has access to richer reward signals. We have extensions running that compose multiple reward categories (a stability term, an eval-trajectory-slope term, a regularization-budget term) into the Sidecar's advantage. Each new category gives the policy more signal to act on; preliminary results show each new category contributing additional gains on top of the 42% baseline.

A modest control-channel set. The Sidecar in the foundational test controlled roughly a dozen distinct channel types. There are many more forward-pass dials that could be added. The per-tensor weight-decay channel described in detail in §5 was added *after* the foundational result

and demonstrates that adding even one additional channel (a single new dial) can produce a further endpoint advantage in the high-teens-percent range. That additional gain is on top of the 42% foundational result, not included in it.

What this means in plain terms: the 42% figure is the result of applying a small, deliberately scoped instance of the paradigm to a deliberately scoped surface. The paradigm's headroom is the product of (number of channels) × (number of reward categories) × (richness of action space within each channel). The foundational experiment exercised only a fraction of each. The thesis is not "we got 42% and we're done"; it is "we got 42% with the minimum scoped version of the approach, and every subsequent extension has produced further gains in the direction the thesis predicts."

We treat the 42% figure carefully for this reason. We do not lead with the larger composite numbers from later, more extended experiments, because those numbers conflate the foundational result with subsequent extensions and obscure what was actually validated. The 42% is the clean test of the hypothesis. Everything beyond it is incremental refinement and extension that builds on the validated foundation.

5. A demonstration of the headroom: per-(layer, tensor) weight decay

The clearest single experiment in the post-foundational extensions added one new control channel to the Sidecar: a per-(layer, tensor) weight-decay coefficient learned end-to-end against the model's own loss. The experiment is worth documenting in detail both because the result is striking (a substantial additional endpoint gain on top of the foundational +42%) and because the *kind* of finding it produced (structural rather than numerical) is exemplary of what the Sidecar paradigm makes possible and what hand-tuning cannot.

5.1 The mechanism in principle

One learnable scalar per (layer, tensor) cell (for an 11–layer body with seven body tensors per layer this is seventy-seven scalars), each initialized at a sensible default, each sampled from a per-cell Gaussian stochastic policy at every training step, each updated by REINFORCE using the next step's loss change as the reward signal. Per-cell baselines absorb the loss-trajectory mean so the advantage measures the marginal effect of *this* cell's perturbation rather than the model-wide direction of travel. The policy has wide latitude: it can shrink a cell's regularization to effectively zero or push it well above any value a human would have considered.

One mechanical choice in the consumer is load-bearing enough to call out: the weight-decay update is *purely multiplicative*. A tensor at zero weight decay is updated exactly as in the baseline; a tensor at high weight decay is shrunk but never rotated. This one-sided action space turns out to be the difference between success and failure for the entire mechanism, and §6 explains why a different design choice on the *same* mechanism architecture failed.

The implementation added approximately seventy-three lines of functional code to the training binary. The small footprint matters: it means the result is reachable inside any training stack whose control plumbing already routes through a Sidecar. Most of the engineering cost of the Sidecar paradigm is paid once when the Sidecar itself is built; each new control channel adds modestly to the surface area.

5.2 What the policy chose

Figure 2 shows the per-(layer, tensor) weight-decay coefficients across a 15K-step training run. Every cell starts at the same initial value. By the end of training the cells have separated across nearly two orders of magnitude. A handful of tensors (attention-key projections in the early-to-middle of the stack, the FFN gate at layer 4, an output projection at layer 8) have been pushed to roughly 5–20× the initial value. A roughly equal number of tensors have been pulled down to the clamp floor. Most tensors stayed near the initial value, which is itself a meaningful choice the policy made.

Figure 2. The Sidecar's per-(layer, tensor) weight-decay choices over 15K training steps. Color is log-scaled; the dynamic range from clamp floor to clamp ceiling is roughly three orders of magnitude. Each row is one of seventy-seven body tensors; each column is a checkpoint. The mid-stack attention-key projection (L03.wk) was driven to roughly 20× the initial value, the single largest regularization choice in the model.

5.3 The headline incremental result

Against an otherwise identical baseline (the Sidecar with the foundational channel set but *without* the per-tensor weight-decay channel), the run with the additional channel was *worse* across every eval point in the first 12K steps. The stochastic perturbations the policy introduces tax early fitting: the model cannot sharp-fit because every step shrinks the weights by a randomized amount, drawn fresh from the per-cell distribution. At step 12K the no-extra-channel baseline reaches its peak (lowest-perplexity) checkpoint. From that point onward the baseline regresses sharply (a roughly +25% perplexity blow-up between step 12K and step 14K), while the new-channel run continues to improve. By the end of training the new-channel run is 19.3% ahead of the no-extra-channel baseline on the primary held-out eval. Figure 3 plots both trajectories.

Figure 3. Held-out eval perplexity for the Sidecar-with-foundational-channels baseline (grey) and the Sidecar-with-added-per-tensor-wd-channel run (green). The added-channel run pays a 7–15% perplexity tax through the first 12K steps in exchange for preventing a late-training regression the baseline suffers. The crossover sits between steps 12K and 13K; the endpoint advantage is 19.3% on top of the foundational +42%.

The advantage is structurally interesting. The added-channel run's endpoint perplexity is approximately equal to the baseline's *peak* perplexity at step 12K. In other words, the new channel is not finding a better optimum. It is finding a *more stable* optimum, one that does not collapse as the baseline does. This is precisely the kind of trajectory shape that a single global weight-decay coefficient cannot produce: late-training instability is by definition concentrated in a small subset of (layer, tensor) cells whose contributions overgrow, and a global coefficient that can suppress those cells must also suppress every other cell, paying the early-training tax across the whole model rather than only where it is needed.

5.4 The counterintuitive finding: the policy detects something static analysis cannot

The closed-loop policy chose to push some (layer, tensor) cells to high regularization and others to near-zero. The obvious question is whether the choices correlate with any property of the tensors themselves: whether, for instance, the heavily regularized tensors are the ones with large weight norms (the ones a competent researcher might intuit as overfitting candidates).

They are not. We computed Pearson correlations between the policy's end-of-training weight-decay choices and several static weight statistics: Frobenius norm, standard deviation, max-absolute weight, sparsity, and the magnitudes of the AdamW first and second moments. The strongest correlation, with Frobenius norm, was $r = -0.20$. The correlation with standard deviation was $r = -0.04$. **No static property of a tensor predicts what the policy will do with it.** Figure 4 makes the point visually.

Figure 4. The learned weight-decay coefficient at end of training plotted against the static Frobenius norm of each (layer, tensor) cell in the no-extra-channel baseline run. Each point is one of seventy-seven body tensors; colors indicate tensor type. The most heavily regularized tensor in the model (an attention-key projection at layer 3) has roughly the smallest Frobenius norm in its tensor family. The correlation across all seventy-seven cells is $r \approx -0.20$, effectively zero. The policy is reading something the static statistics cannot show.

The single most heavily regularized tensor in the model has roughly the *smallest* Frobenius norm in its tensor family. Attention-value projections at the same layer indices, with similar Frobenius norms, were left alone. FFN gate projections with five times the max-absolute weight of the

heavily-regularized cells were left near baseline. Whatever signal the policy is detecting, it is not weight magnitude.

We interpret this as follows. On each training step the policy samples a perturbation for every cell, observes the resulting change in loss, and over many samples builds an estimate of the marginal sensitivity of loss to weight-decay perturbation at that cell. That sensitivity is fundamentally a closed-loop, in-context quantity: it depends on the rest of the network's current state, on the data distribution being learned, on the gradient flow patterns through the cell, and on the way the cell's contribution interacts with downstream layers. It is not a function of the weights alone. Static analysis cannot surface it because it does not exist outside the loop.

This is, to our reading, the central scientific claim that comes out of the broader research line. The reason hand-tuning has a ceiling is not that humans pick poorly. It is that the right value of a per-tensor regularization knob is not readable from the tensor. It is a property of the loss surface, the data, and the rest of the network, jointly. The only mechanism that can read it is a closed-loop one. Every additional Sidecar-controlled channel we add inherits this property: the policy reads a closed-loop sensitivity that no offline analysis can recover, and the corresponding training gains compound.

5.5 Downstream weight forensics

The mechanism's effect on the final weights is large and structured. Comparing the body weights of the new-channel run against the no-extra-channel baseline at end of training, the average cosine similarity per tensor is 0.88; none of the seventy-seven tensors are above 0.95. The mean $\|\Delta w\| / \|w\|$ is 0.49: the weight vectors are, on average, *half* as far apart as they are long. The two runs produced visibly different bodies, not two near-identical bodies separated by a small regularization term.

The pattern in the body diff is the one Figure 2 would predict. The heavily-regularized cells shrank substantially: the L03 attention-key projection lost ~47% of its Frobenius norm and its peak weight collapsed from 0.107 to 0.075; the L04 FFN gate's peak weight collapsed from 0.352 to 0.056, an 84% reduction. The most divergent tensors in cosine space, however, are not the heavily-regularized ones. They are the FFN down-projections, which the policy left alone. The down-projection sits downstream of every regularized cell and absorbs the integral of every upstream change. This is the signature of a mechanism that operates locally on a small subset of cells but produces global effects through the network's gradient cascade. (The downstream forensics methodology, effective rank, per-tensor cosine, role classification, is the subject of the fourth paper in this series.)

6. The principle generalizes: early backward-pass extensions

The per-tensor weight-decay result is one of several extensions running the same Sidecar playbook against different training-loop dials. As an early probe of the backward-pass control surface, we ran a coordinated batch of four variants, each turning one backward-pass operation into a Sidecar-controlled policy:

- **Per-layer gradient skipping.** A Bernoulli policy per layer; with the sampled probability, zero out all body gradients for that layer for that step before the optimizer update.
- **Per-tensor gradient gating.** A per-layer probability used independently for each of the seven body tensors, zeroing each tensor's gradient with that probability.
- **Per-step contiguous gradient truncation.** A scalar per layer averaged across the stack and quantized to an integer K, then used to zero gradients on the first K layers.
- **Gradient-momentum blending.** A per-layer mixing coefficient that blends the raw gradient with the optimizer's first-moment estimate before applying the optimizer step.

Of the four, none beat the foundational Sidecar baseline cleanly on the primary eval. Two regressed mildly through middle training and recovered to near-parity; two regressed sharply and were stopped early. This was the first batch of backward-pass experiments and the result clarified rather than discouraged: extending the Sidecar to the backward pass is a real research problem, with its own design challenges, not a free additional gain.

The successful weight-decay channel (a forward-pass control) and the four less-successful backward-pass variants together tell a sharper story than either would in isolation. **The action space matters enormously.** Weight decay's action (multiply a weight by a number between zero and one) is *one-sided*. The worst the policy can do, locally, is freeze a tensor. The gradient-mixing action is *two-sided*: blending the raw gradient with the optimizer's first moment can push the effective update in any direction in the gradient's span, and the policy gradient's local-Taylor pull is toward whichever direction maximizes one-step reward, regardless of whether that direction is a long-run improvement or a long-run trap. We observed, repeatedly, the latter: REINFORCE finds local-Taylor attractors that maximize next-step reward while accumulating compounding harm. One run blew up at step 10K with a 2,100% perplexity regression after looking fine through 8K.

The principle we extracted: **when designing a new Sidecar-controlled dial, prefer action spaces whose worst-case effect is bounded.** A multiplicative shrink-only update has bounded worst-case. A clip-only threshold has bounded worst-case. A skip-only gate has

bounded worst-case. Anything that can rotate or amplify a gradient does not, and the policy will find the rotation that hurts you. This is a design rule we now apply to every new control channel, forward-pass or backward-pass.

The other lesson from that batch was about reading defaults. When a control channel emits a scalar that is interpreted downstream as a probability, the natural emission baseline of the channel matters enormously. Several of our backward-pass channels emit values centered near ~1.0 rather than near zero, because they share the underlying instruction set with other channels whose semantic neutral value is one. Interpreting that raw emission as a Bernoulli probability without subtracting the natural baseline produces a ~60% skipping rate from step zero: a catastrophic prior. The lesson generalizes: **always know what "neutral" looks like in the emission space before composing a new control on top of a shared channel, and subtract that baseline explicitly in the consumer.** This is the kind of integration bug that does not surface in unit tests, because the channel is producing correct values; the consumer is reading them at the wrong zero.

The backward-pass research line continues. The early variants ruled out four design choices, surfaced two load-bearing design rules, and pointed at the next set of architectures to try. The backward-pass headroom remains, in our view, larger than the forward-pass headroom: the dials are more numerous, the effects on training dynamics are more direct, and the Sidecar's existing reward-path machinery extends to backward-pass control with modest engineering. We expect the next year of work in this line to deliver further substantial gains on top of the +42% foundational figure.

7. A pre-Sidecar discovery: per-layer learned learning rates

The earliest validation of the Sidecar architecture, predating the formal foundational experiment, was a per-layer learning-rate multiplier: a single control channel that emits one scalar per body layer, with the optimizer's update for every parameter in that layer scaled by that scalar before being applied. This channel has been load-bearing in the production substrate for over a year.

The most informative single run from that early line was an experiment we will call the long-horizon variant. The variant was launched with a generous clamp range on the per-layer LR scalar; over the course of training, REINFORCE drove all eleven body-LR multipliers down to the clamp floor by approximately step 3K-5K. The body weights effectively *froze* from that point

on. The bridges between layers, the embeddings, and the language-model head continued to train normally. The final result outperformed a same-architecture baseline with a uniform learning rate by 12% on the primary eval, at a 20% smaller parameter count.

We did not design this strategy. We did not anticipate it. The Sidecar discovered it.

Figure 5 shows the per-layer learning-rate trajectories from a longer downstream run with the per-layer LR channel active. The layers separate visibly over training; the highest-LR layer at steady state is operating at roughly three times the lowest, and the relative ordering shifts over training as the model's optimization needs evolve.

Figure 5. The per-layer learning-rate scalar emitted by the Sidecar across a 102K-step training trajectory. Each line is one body layer; the policy has learned a distinct trajectory for each. The relative ordering shifts over training as the model's per-layer optimization needs change. None of these schedules were specified by a human.

The discovery itself was the result. The original variant's implementation has since been rebuilt with a cleaner policy parameterization: the freeze-the-body schedule it discovered is reproducible from the cleaner build, and "expose plumbing and let the policy find schedules" is now the standard operating mode for every new control channel we wire. The lesson generalizes: a useful schedule discovered by an early implementation is not invalidated by the implementation being rebuilt; the schedule is the signal that the reward path was rich enough to surface non-obvious strategies, and that property is what we are after.

The freeze-the-body schedule has also stood up as a hypothesis-generator. We did not know, going in, that freezing the body of a transformer mid-training and continuing to train only the boundary modules was a competitive strategy. We *do* know now, because the policy demonstrated it. The body of work that followed has explored why (what specifically the boundary modules learn when the body is held still, what depths of body matter most for downstream task performance, how the schedule interacts with regularization), and several of those follow-up investigations have produced their own findings. None of that follow-up work would have been generated without the original discovery, and the original discovery would not have happened without the Sidecar paradigm.

8. Design rules

The reward-driven training-control line has converged on a small number of load-bearing rules. We treat them as the design discipline that distinguishes a control surface that works from one that does not.

Never hard-code a hyperparameter that can be learned. Every Adam call, every gradient clip, every regularization coefficient, every threshold should ride a Sidecar-emitted scalar. The default value of that scalar should be whatever value you would otherwise have hard-coded; the difference is that the policy can now move it from the default if doing so reduces loss. There is no run where this costs anything material in compute, and there are many runs where it has revealed a non-trivial schedule the researcher would not have written.

Initialize caps dormant. When introducing a cap-style mechanism (a spectral-norm clamp, a state-magnitude limit, a gradient clip threshold), initialize the cap at a value high enough that it does not bind on healthy training. Let the policy pull it downward as evidence accumulates that the cap is needed. We have a clean negative result on this rule: a cap initialized at 200 that needed to be at roughly 10,000 to remain dormant during healthy training cost the run 14% on the primary eval at the 1K-step checkpoint. Aggressive default caps are not a "safety margin." They are an active impediment to early fitting, and the policy will not finish moving them out of the way before the damage is done.

Know the neutral of every control channel. When a Sidecar-emitted scalar is composed into a downstream consumer (particularly a consumer that interprets it as a probability or a small-offset modulator), the natural emission baseline of the channel matters. Shared control channels in our stack emit values centered around ~1.0, not around zero, because they share an instruction set with multiplicative-scaling channels. Consumers must subtract the neutral explicitly. We have one expensive incident on this rule (Section 6) and have since added a per-channel `neutral` field that consumers are required to read.

Verify silent reward-path corruption end-to-end. A single aliased pointer in a backward-pass CUDA kernel (a `+=` accumulation where two pointer arguments happened to alias to the same buffer) silently doubled gradient flow through every layer of one of our recent builds, producing a ~2,000× amplification across the body's depth. The forward loss looked fine; gradient clipping absorbed the visible damage; the model trained badly but did not crash. The bug was found only by end-to-end tracing of a specific reward-path code change. The rule we extracted: **reward-path code needs end-to-end traces from controller emission to weight update, not unit tests at each layer.** Unit tests cannot catch a corruption that flows through

multiple layers of indirection and only manifests as "the model trains, but worse than it should." The trace must follow the actual numerical path of a single sampled action all the way to the final weight update on a real step.

Don't claim novelty for plumbing that already exists. This is a process rule rather than a design rule, but it is load-bearing for our own velocity. Several of our most useful control channels (per-layer learning rate, per-layer gradient scale, per-tensor weight decay) were wired in successive earlier substrate versions and have been quietly available to every experiment since. Before proposing a "new" knob, check whether the existing Sidecar already exposes it; if so, the work is in the consumer code, not the controller.

9. Where this is going

The foundational +42% result validated the paradigm. The work that follows that result is the project of cashing the validation in: extending the Sidecar's control surface, enriching its reward signal, and removing the constraints that bounded the foundational test. Three directions are active.

Extending to the backward pass. This is the largest single source of remaining headroom. The dials are numerous (gradient routing, optimizer-state hyperparameters, momentum schedules, projection blends, per-tensor learning-rate scales independent of the per-layer scales already active), the effects on training dynamics are direct, and the design rules from §8 (bounded action spaces, dormant caps, neutral-aware composition) give us the principles for designing channels that won't destabilize training the way the early backward-pass variants did. The next year of work in this line is heavily weighted toward this surface.

Enriching the reward signal. The foundational Sidecar learned against a single reward: loss delta. The policy's ability to discriminate among cell-level interventions is bounded by how informative that reward is at the cell level, and a single global loss signal is a coarse measure when the policy has seventy-seven cells to update. Composite reward signals (stability terms, eval-slope terms, regularization-budget terms, late-training-divergence terms) give the policy more information per step, and each new term we have integrated has produced further gains. The research project here is figuring out which composite rewards are informative without being adversarial to one another.

Letting the model decide its own architecture. Once the training-control layer is mature, the natural next question is whether the same paradigm extends *upstream* of the training loop, to the architecture itself. If the Sidecar can decide how each weight should be updated, can a similar policy decide what shape the weights should take? This is the subject of the next paper

in the series (*"Self-Discovering Architectures"*). The architectural extension inherits the Sidecar's discipline (REINFORCE on a stochastic policy with learnable variance) and runs into a new class of problems (mid-training tensor surgery, optimizer-state coupling, reward-window calibration) that the training-control work did not encounter. The diagnostic methodology is now mature; the headline competitive result remains open.

10. What you should take away

The single most important claim in this paper is also the simplest. **A learned policy operating on top of training, given the right plumbing and the right reward, finds better hyperparameter trajectories than a human researcher can hand-pick.** We tested this hypothesis with the deliberately scoped foundational experiment of §3 (Sidecar versus vanilla, forward-pass-only control, single reward signal, matched everything else), and the result was a 42% improvement on the primary held-out eval at matched compute. The paradigm is validated.

What we *do* with that validated paradigm is the project that follows. Adding control channels delivers further gains (§5: a per-tensor weight-decay channel alone added 19.3% endpoint advantage on top of the +42% foundation). Extending the reward signal delivers further gains. Extending to the backward pass (the largest remaining source of headroom) is the active research front. And the same discipline, applied upstream of the training loop to the architecture itself, generalizes to a new class of problems documented in the next paper.

The 42% figure is not the final number. It is the entry-level validation of a paradigm whose ceiling we have not yet found, on a control surface we have not yet finished mapping, with a reward signal we are still enriching. Every extension we have built on top of it has produced gains in the direction the thesis predicts. That cumulative pattern, more than any single number, is the evidence we offer that the Sidecar paradigm is sound, generative, and the right foundation for the work that follows in this series.

This is Paper 2 of a four-paper series. Paper 1 ("Research Posture: Why We Let the Model Choose") sets the methodology this work uses; Paper 3 ("Self-Discovering Architectures") extends the Sidecar paradigm upstream of the training loop to architecture itself; Paper 4 ("Emergent Layer Roles and Functional Specialization") develops the forensic framework we use to read what each Sidecar-driven intervention did to the body of the trained model.