PROFORMA GLOBAL RESEARCH

Research Posture: Why We Let the Model Choose

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-05-18

Executive summary. The firm's research program is organized around a single thesis: the right way to make any decision inside training (every hyperparameter, every threshold, every gate, every architectural commitment) is to let a reinforcement-learning policy make it, not a human picking a value. Human-picked values are bootstrap only. The mature instances of any decision class in our stack are scalars learned by the same reward signal that trains the model itself. This paper documents the operational discipline that follows from that thesis: the Fire-Aim-Fire iteration loop, the post-step retrospective, the append-only findings ledger, and a research culture in which negative results are first-class artifacts. The thesis has been tested. The foundational experiment, documented in the next paper, attached a small reinforcement-learning policy (the *REINFORCE Sidecar*) to the forward-pass control surface of a standard training loop and beat an otherwise-identical hand-tuned vanilla baseline by 42% on the primary held-out evaluation at matched compute. That result anchors everything that follows in the series: the Sidecar paradigm extended to additional control surfaces, the same primitive applied upstream to the model's own architecture, and the forensic framework that reads what each rung of the progression did to the body of the trained model. The three technical papers that follow document those extensions; this paper exists to explain why they look the way they do.

1. The thesis: hyperparameters are a tax on the data

Modern transformer training is held together by hundreds of numbers a human chose. Learning rate. Weight decay. Gradient clip. Attention head count. Hidden width. The temperature on every softmax. The epsilon in every Adam denominator. The clamp on every saturating function. Every one of these encodes a prior: a guess about what regime the training will land in and what the right response to it is.

That guess is wrong almost everywhere. A learning rate sized for early training is too high after the loss landscape sharpens. A gradient clip tight enough to suppress an exploding step is tight enough to throttle a healthy one. A hidden width chosen uniformly across depth is a statement that the data has no preferred depth structure, which is empirically false for every domain we have measured.

The standard response is hyperparameter search. Run a sweep. Pick the best. Lock it in. But sweeps deliver a single static configuration optimized for an average over training, and training is not stationary. The configuration that produces the best final checkpoint is not the configuration that should have been used at step 1, step 1000, or step 100000. A scalar trained against the same reward signal that trains the weights does not have this problem.

Our thesis is therefore simple and absolute: **every hyperparameter, threshold, gate, and routing decision should be learned, not hand-picked.** Human-picked values are bootstrap only. Once the path exists for the model's own reward signal to flow into a decision, the human's number is replaced by a learnable scalar, and the human's authority over that knob is permanently transferred to the model.

This is not a methodological flourish. It is the central engineering commitment of the firm's research program, and every subsequent decision (what to build, what to instrument, what to publish, what to revisit) follows from it. The commitment has been tested, and the foundational test is documented in the next paper: an otherwise-identical training run with a learned policy on top of the forward-pass control surface beat a hand-tuned vanilla baseline by 42% on the primary held-out evaluation at matched compute. The 42% number anchors everything that follows. It is the result that licenses us to extend the same paradigm to additional control surfaces, additional reward signals, and ultimately to the model's architecture itself.

2. Knob-ownership migration

The clearest way to see the posture in practice is to look at the trajectory of a single project's hyperparameters over its lifetime. At the start, humans chose everything: the global learning rate, weight decay, layer count, hidden width, head count, gradient clip, every cap and threshold. Today, on the active substrate, the following are learned per (layer, tensor) by reward-driven updates:

- per-layer learning rate scales (each weight matrix's effective LR rides a REINFORCE-driven gain)
- per-tensor weight decay
- per-layer hidden width and intermediate width (mutated mid-training by a Gaussian policy with learnable mean and learnable variance per axis)
- backward-pass behavior: gradient skip schedules, truncation depths, and projection directions

And the following are wired and undergoing iteration. The plumbing exists; the policies have not yet outperformed their hand-set defaults:

- per-layer attention head count and head dimension (extension of the architectural-mutation policy class to head axes: the first attempt failed because the reward signal could not see the higher recovery cost of head-axis surgery; a per-axis reward window is in flight)
- the exploration magnitude on the architectural-mutation policies (σ is learned per axis alongside μ; an analogous σ-learnability path is being added to the older sidecar policies, where σ is still a hand-set constant in several places)

The reverse direction (taking a knob away from the model and giving it back to a human) has happened zero times. Once a parameter becomes learnable, it stays learnable. The trajectory is one-way.

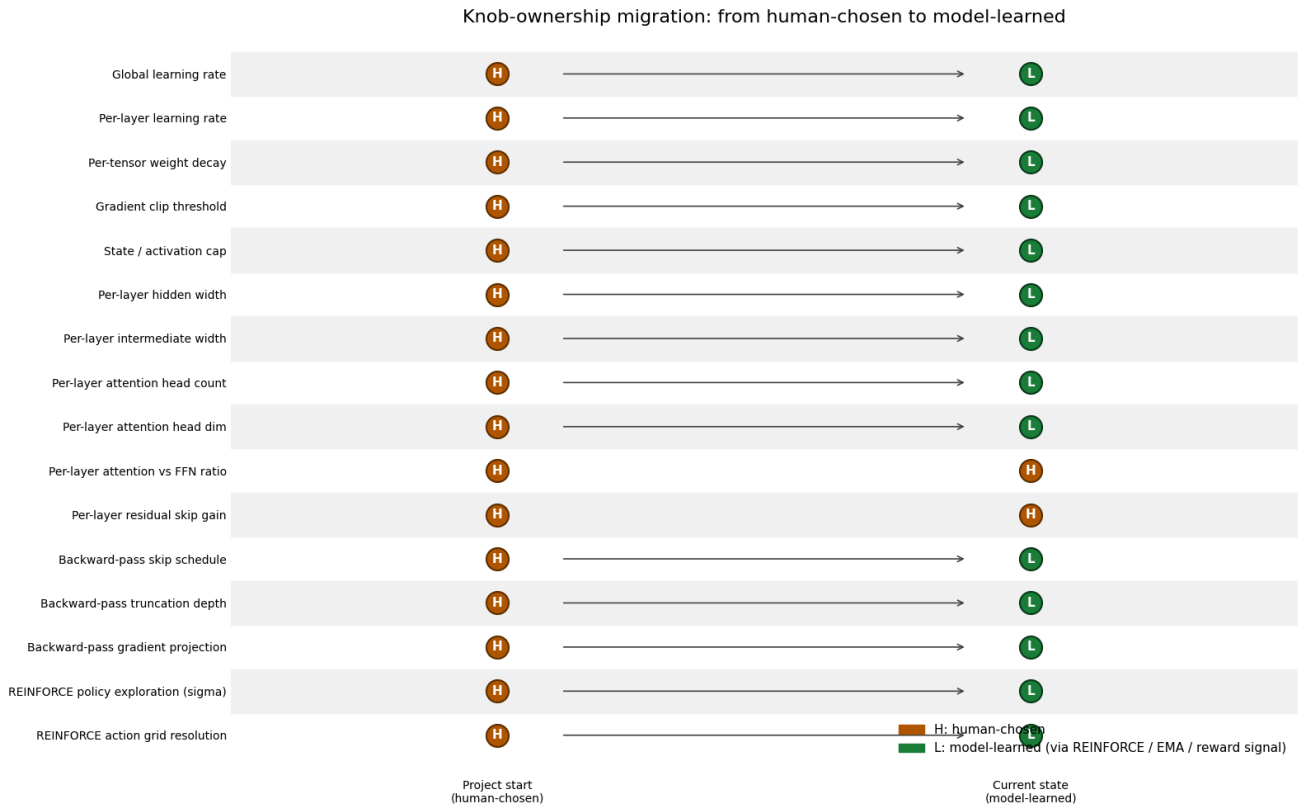

Figure 1. Knob-ownership migration over the project's lifetime. The arrows are the work.

The chart understates the change. The "global learning rate" row, for example, does not capture that the current substrate has roughly one learnable LR scalar per (layer, weight matrix): a single migration row stands in for hundreds of individual scalars whose values are now set by the data,

not by us. The remaining human-chosen rows are not stable preferences. They are the next migrations on the queue, sequenced by which reward path is cheapest to build next.

3. Expose plumbing, never decide for the model

The principle that drives migration is operationally captured in a single rule: **expose plumbing, never decide for the model.** When we add a feature, we add the path by which the model can express a decision; we do not encode the decision itself. The discipline applies in two directions.

The first direction is replacement. When we encounter a hand-picked number in existing code (a clamp, a blend weight, a temperature), we ask whether the model has any way to change it based on experience. If the answer is no, we wire one. The number becomes a learnable scalar, the reward signal becomes an update rule, and the original value becomes an initializer rather than a constant. The migration is permanent.

The second direction is construction. When we build something new, we expose ideas (observables, optional routing branches, gates that can be opened or closed) rather than prescribing what should happen with them. A canonical example: rather than fixing a state-norm cap at a value chosen to "feel safe", we expose a learnable threshold per layer, initialize it dormant (high enough that healthy training never triggers it), and wire a reward-driven path that pulls it down only if the model decides the post-projection state is more useful than the pre-projection state. The cap exists. The cap's value is the model's to choose.

The dormant-by-default pattern is load-bearing and earns its own treatment. Early in the project, a new cap was shipped pre-tuned to a value chosen by hand to be "tight enough to catch the explosion we feared." A controlled comparison against the same code with the cap initialized at 50× that value (effectively disabled) showed the pre-tuned version was 14% worse on the primary training metric at the same compute budget. The cap was actively impeding training before any explosion occurred. The lesson: a hand-picked cap value is the same mistake as any hand-picked threshold. The cap mechanism is useful; the cap *value* should be initialized so it does not bind, and a reward path should own its downward adjustment.

The same logic applies to every gate we have ever added. A gate initialized at the value the researcher believes is correct is a gate whose value has already been chosen. A gate initialized in its dormant configuration, with a reward path that can pull it open, lets the model tell us whether the gate should exist at all.

4. Fire → Aim → Fire

The migration trajectory described above is not free. Each new learnable scalar requires that the reward signal actually reach it through a non-degenerate gradient path, that the policy's own hyperparameters be sensible enough not to collapse exploration, and that the resulting telemetry let us verify the lever is doing what we believe it is doing. This is where the iteration discipline comes in.

We call the loop Fire-Aim-Fire, and it is more carefully scoped than the name suggests. Fire-Aim-Fire is not a build-order prioritization rule. It does not mean "ship one feature, then start the next." Multiple features can be in flight at any time. What the loop governs is the obligation that attaches to a feature once it has been shipped: **before declaring it useful, you must instrument it deeply enough that the data can render a verdict.**

The three phases:

1. **Fire.** Ship a minimum viable version of the feature on a fresh fork. Do not gold-plate it; do not pre-optimize. The goal is to get the path running end-to-end so the reward signal can begin moving against it.
2. **Aim.** Add the observability needed for the model to tell us if the feature is garbage or gold. This is the phase that gets cut under time pressure and that we have explicit standing rules against cutting. If a feature ships without the telemetry to verify it, the feature has not shipped; it has been deployed.
3. **Fire (again).** Iterate based on what the telemetry says. Keep, kill, or tweak. A feature that survives this loop has been *proven* useful against a measurement we trust. A feature that does not survive enters the findings ledger as a constrained region of the search space.

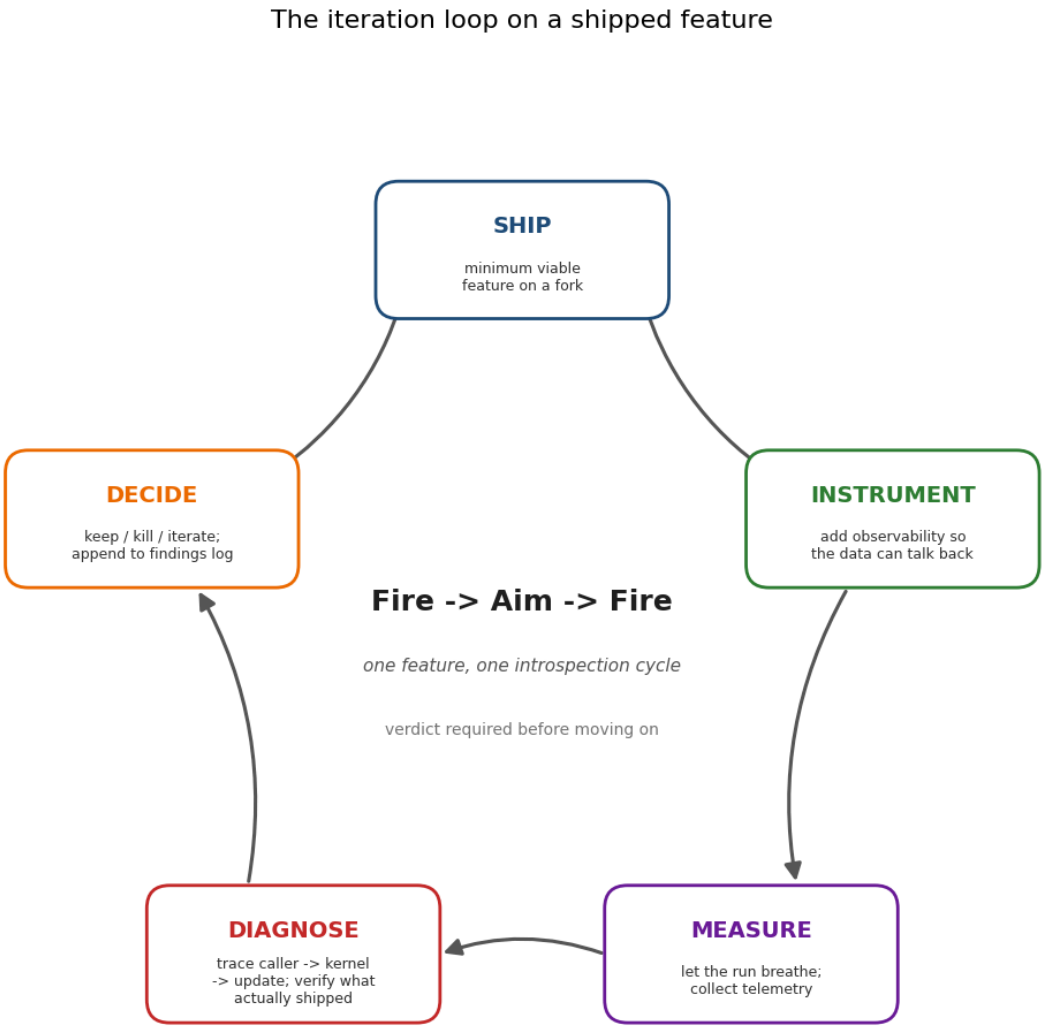

Figure 2. The iteration cycle on a single shipped feature. Multiple features run the cycle in parallel; what matters is that each one completes it.

Two operational consequences flow from this scoping. First, "we shipped X" and "we verified X works" are different claims, and we have learned to be ruthless about not eliding them. A prior release in this project shipped a multi-lever feature whose deploy header confidently announced four new control paths. A source-level audit later revealed that two of the four wrappers were aliases of a single underlying function, one of the underlying calls was a stub, and the net effect was that gradient buffers were being multiplicatively passed through the same scale four times per step. The metric regression was real and lasted across several

subsequent releases, each of which trusted the original deploy header and chased the wrong cause. The standing rule that emerged: before declaring a multi-lever feature working, trace the full caller → dispatch → kernel → update path in code and show the trace before claiming the feature is live. Headers lie. Source does not.

Second, the introspection phase is where most of the work happens. The model's training process is the diagnostic instrument; our job is to expose enough state to read it. When a feature appears not to work, the first hypothesis is not that the feature is bad: it is that we are not yet measuring the right thing. A feature without observability cannot be evaluated, and an unevaluated feature cannot be killed responsibly. Both directions of the verdict (kept and killed) require the same instrumentation budget.

5. The post-step retrospective

The iteration loop is supported by a smaller, faster discipline applied after every meaningful step of work. Three questions, every time:

1. **Did this step actually work the way we wanted?** Compare what happened to what was predicted. If they diverge, the divergence is the most valuable data of the step. Do not paper over it.
2. **Are we using the right substrate, or routing around it?** When a fix or a new feature requires bypassing the project's existing infrastructure ("the proper mechanism is broken so let me edit in place"), the bypass is almost always the wrong move. The proper mechanism is load-bearing for reasons that may not be visible from the current task. If the proper mechanism is broken, fix the proper mechanism; do not establish a precedent for bypassing it.
3. **Is the next action actually necessary, or are we manufacturing work?** A great deal of researcher activity is defensive completeness: running one more analysis, dumping one more CSV, building one more heatmap to "see what it shows." Before any such analysis, write down the decision it serves. If two outcomes of the analysis would lead to the same architectural change, the analysis is not actionable, and the time is better spent on a probe that does branch behavior.

The third question deserves an aside, because it cuts against a common research instinct. In a prior session we proposed a tensor-level attribution analysis at fine granularity: twelve numbers per layer across twenty-four layers. The analysis was technically possible and would have produced a striking heatmap. It was correctly killed before being run, because the architectural insertions the analysis was supposed to inform were already determined by a much coarser grouping (attention versus feed-forward), and the fine-grained version would have led to the

same decision either way. The right aggregation level for an analysis is the level at which the answer changes the next action. Two-hundred-and-eighty-eight-cell heatmaps that resolve to "same as the three-bucket version" are a credibility risk: they suggest more rigor than they deliver.

6. The findings ledger and the memory system

A research program of this kind generates lessons faster than any single researcher can hold them. Worse, lessons learned in one session evaporate by the next unless they are written down in a form the next session will actually read. We treat this as an engineering problem and solve it with two layered artifacts.

The first is an **append-only findings ledger**. Every material observation from a training run (a confirmed bug, an unexpected convergence pattern, a discovered structural commitment by the model, a hypothesis that survived or died) gets a dated entry. The format is strict: a timestamp, a short headline, and sections for what was confirmed broken, what was confirmed working, what surprised us, and what the action items are. Nothing is overwritten. If a later finding revises an earlier one, the revision is a new dated entry that cites the prior one, not an edit to the old one. The ledger's job is to make it impossible for a future session to re-derive the same lesson from the same logs and call it a discovery.

The second is the **promoted memory file**. A finding that survives a few iterations and is durable enough to function as a design rule (not merely a data point about a specific run) is promoted to a named memory file that loads on every session. These files are short, prescriptive, and dated. They carry incident citations: the specific failure that justified the rule, so a future session that is tempted to violate the rule can see exactly which historical mistake it would be recreating.

The combination matters. Without the ledger, the memory files would have no provenance. Without the memory files, the ledger would be a graveyard of lessons no future session reads. Together they constitute the institutional memory of the project, and they are why the lessons of this paper compound rather than dissipate.

7. Honest failure analysis is the strongest credibility signal

Most ML research labs publish wins. Negative results are written up reluctantly, if at all, and are typically buried inside an appendix. This is methodologically backwards. **Negative results constrain the search space.** A confirmed dead end is, from the perspective of any subsequent

researcher, more valuable than yet another paper claiming a modest gain on a contested benchmark: the dead end is a region that no longer needs visiting, and the constraint compounds across the field.

We treat negative results as first-class outputs. Three illustrative examples from this project, paraphrased to respect the disclosure boundary:

Hand-picked variable-width architectures lose to uniform at matched parameter counts. A multi-week sequence of experiments built variable-width transformer specs (narrowing or widening hidden dimensions across depth according to several plausible templates) and compared them against a uniform-width baseline at the same total parameter count and the same compute. The variable-width specs lost on convergence speed and on per-step throughput. The throughput loss had a specific cause (tensor-core alignment penalties on non-standard dimensions); the convergence loss did not have a single cause and was not recovered by tuning. The conclusion was not "uniform is better full stop"; it was "uniform is better than any hand-picked variable shape we tried, which means the next thing to try is letting the model pick the shape." That subsequent experiment, conducted with a learned per-layer width policy, converged to a roughly 3.6× sawtooth profile that no human spec had proposed, and *also* underperformed the uniform baseline at matched compute. The forensics on *why* it underperformed turned out to be the actual value of the experiment: mid-training architectural surgery was lobotomizing the optimizer's hidden state, and we would not have known to look without the negative result. The positive result is the methodology and the diagnostic, not the shape. Both results (the failure of hand-picked specs and the deeper failure of the first learned-policy attempt) are part of the same finding, and publishing only a sanitized version of either would misrepresent the trajectory.

Hardcoded exploration magnitude in a reward-driven policy is an anti-pattern. An early REINFORCE-style policy in this project shipped with a fixed exploration parameter: the standard deviation of the Gaussian noise added to each action. The fixed value was chosen by hand at what seemed a reasonable order of magnitude. It was too aggressive for the signal regime the policy was operating in, the policy could not compensate (the standard deviation was not on its gradient), and the policy never converged to a useful behavior. The fix was to make both the action mean *and* the action standard deviation learnable per granular unit, with the standard expressions for the REINFORCE gradients of each. The deeper lesson, which now sits in a memory file: a reward-driven policy must learn *every* hyperparameter that materially affects its behavior, not just its action mean. The exploration magnitude is the most commonly missed; in our practice it is now non-negotiable.

Aggressive default caps actively impede early training and should be initialized dormant. Already discussed in Section 3. The negative result quantified the cost (14% worse on the primary metric at the same compute), and the resulting design rule ("initialize caps so they do not fire during healthy training; let observation and reward drive them downward when needed") is now a load-bearing convention applied to every new cap-style mechanism in the project.

Each of these is a published-as-internal-memory loss that compounds into a constraint on what we will and will not try next. The cumulative effect is that a researcher joining this project today inherits not just the current code but a map of regions that have already been searched and ruled out. That map is, in our view, the most underrated artifact of any research program.

8. What this posture costs and what it buys

We owe a frank accounting of the tradeoffs. Letting the model own a knob is more expensive than picking a value. The reward path has to be wired, the policy needs its own learnable hyperparameters, the telemetry has to be instrumented, and the variance in early training is higher than a tuned baseline because the policy is exploring rather than executing. A research culture optimized for fast headline numbers would not make these choices.

The compensation is that every learned knob is mostly self-solving from that point on. We tune it less, and we tune it differently: the work moves from picking values to picking initializations, action spaces, and reward shapes. The reward signal handles regime shifts as training progresses, and the same code path adapts to a different model size, a different dataset, or a different compute budget without re-sweeping. Hand-tuned hyperparameters are a recurring liability; learned hyperparameters are a one-time-and-occasional-revisit engineering investment.

The compounding is the point. Each migration of a knob from human to model frees the researcher to work on the next migration. The architectural changes documented in the technical papers that follow (dynamic per-layer width and head-count mutation, reward-driven backward-pass control, cross-layer routing structures discovered rather than specified) are reachable only because the cumulative effect of prior migrations has freed enough researcher attention to build them.

9. How to read the technical papers

The three technical papers that follow describe specific systems built under the posture documented here. They form a deliberate progression rather than three parallel threads, and the order matters.

Paper 2: ***Reward-Driven Training Control*** documents the foundational result: the REINFORCE Sidecar paradigm, the +42% over vanilla at matched compute, the constraints on that result (forward-pass-only, scoped control-channel set, single reward signal), and a worked example of an additional control channel (per-(layer, tensor) learned weight decay) that demonstrates the kind of incremental gain that becomes available once the paradigm is in place. Read it first if you are evaluating the technical credibility of the research program. The 42% number is the entry-level validation of the entire research line.

Paper 3: ***Self-Discovering Architectures*** documents the extension of the Sidecar paradigm one level up the stack: instead of learning the dials inside the training loop, the same primitive (Gaussian REINFORCE with learnable variance) is applied to the model's own architectural shape. The substrate works; the diagnostic methodology is mature; the headline competitive result is not yet in hand because mid-training architectural surgery exposes an optimizer-state coupling that the training-control work did not encounter. We document this honestly because the methodology is the durable output and the competitive number will follow it.

Paper 4: ***Emergent Layer Roles and Functional Specialization*** is the forensic capstone. Each rung of the progression (vanilla baseline, Sidecar with foundational channels, Sidecar with additional channels, architectural mutation) changes the body of the trained model in measurable ways. This paper develops the methodology for reading those changes. The interpretability framework is general; it applies to any decoder-only transformer and requires nothing beyond a finished checkpoint and a brief gradient trace.

The technical papers will be terser than this one, because they assume the posture documented here:

- They will not justify, for each lever, why the lever is learned rather than chosen. The default is learned.
- They will not apologize for negative results. Where an approach was ruled out, the writeup will name what was ruled out, by what evidence, and what was tried next.
- They will not claim a feature works without showing the path by which the claim was verified. Deploy headers do not constitute verification.

- They will be specific about what the model chose. The most interesting result in each system is rarely the headline metric; it is the shape of the configuration the model converged to under reward pressure, which is reliably a shape no human researcher would have proposed.

The reader's takeaway from this paper should not be "Proforma uses learned hyperparameters." It should be that we treat every hand-picked number as a temporary scaffold around a missing reward path, and that the research program is organized around the project of removing those scaffolds one at a time, deliberately, with the discipline to verify each removal and the institutional memory to keep the lessons.

We let the model choose. We build the paths that make choosing possible. The rest is bookkeeping.
