DIFFERENTIATE THE EVALUATOR: CO-SEARCHING PROGRAMS AND PARAMETERS WITH EXACT GRADIENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Scientific models can be expressed as executable programs whose code represents mechanisms, equations, control flow, and numerical procedures. An AI agent may therefore propose the correct scientific mechanism yet have it rejected because its continuous constants are poorly calibrated. In program search, calibration is not merely post-processing: it helps determine which scientific ideas survive. Language models and gradient-based optimization offer complementary strengths, with language models proposing discrete program structure and gradients efficiently calibrating continuous constants. Existing approaches, however, either restrict candidates to fixed differentiable templates or require new tracing, compilation, or implementation work for each proposed program. We instead differentiate the evaluator once, allowing programs supplied as data to inherit exact reverse-mode gradients with respect to embedded constants. Our native differentiable virtual machine executes these programs at runtime and batches many parameter candidates through a single structural execution, making gradient calibration practical as the inner loop of AlphaEvolve-style evolutionary program search. We evaluate the method by co-searching nonlinear WENO shock-capturing schemes for computational fluid dynamics. From language-model-proposed programs, the system reliably recovers the expert-designed WENO-Z+ mechanism and recalibrates its parameters through full solver rollouts. The resulting schemes reduce error by 34–46% relative to WENO-Z on three unseen problems (32–41% against independent reference solutions) and, for one seed, by 26–47% relative to published WENO-Z+, while retaining fifth-order accuracy; a stronger shock test exposes a small undershoot that WENO-Z avoids. Ablations show that evolutionary search identifies the mechanism, while exact-gradient calibration produces most of the cross-problem improvement. Discrete structure also acts as a regularizer: given the same gradient freedom, a strong expert-designed family crosses a stability cliff and develops large shock oscillations, whereas every discovered program keeps a fixed exponent that stays clear of it. A second co-search over state-space programs exercises loops, branches, and matrix operations, and an accompanying optimizer study shows that gradients become increasingly important as parameter dimensionality grows. These results suggest that differentiating the evaluator rather than each candidate program enables AI-driven co-search over discrete scientific structure and continuous parameters without per-program differentiation or engineering effort.

1 INTRODUCTION

AI systems increasingly propose scientific models as executable programs. Language-model searches write candidate algorithms directly as code (Romera-Paredes et al., 2024; Novikov et al., 2025), while symbolic regression returns equations (Cranmer, 2023; Udrescu & Tegmark, 2020). A proposed program specifies both a structure and numerical constants: for example, a decay law and its rate, or a numerical solver and its coefficients. A useful structure can perform poorly with the wrong constants. Evaluating a proposal therefore often requires fitting its constants before deciding whether to retain it.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

For a fixed differentiable model, this is ordinary parameter optimization. Program search adds a complication: each proposal can change the computation itself, including its control flow and the number of parameters. Scientific generative agents combine language-model proposals with a differentiable inner optimizer (Ma et al., 2024); their simulator supplies the gradients needed to fit each hypothesis. We study how to provide that fitting service when the proposed program itself is the computation to execute. Reimplementing each proposal in a differentiable framework is inconvenient, while automatic compilation introduces a preparation cost that may or may not be repaid during a short fit. An interpreter can instead accept each new program as data and reuse the same execution and differentiation machinery.

Our approach is to differentiate the evaluator once. The native differentiable virtual machine (NDVM) reads a program, executes its numerical operations, and records the information needed to differentiate its output with respect to its constants. A new program in the supported language requires no candidate-specific derivative code or native compilation. Its discrete structure remains fixed during fitting; only its continuous parameters change. Unlike interpreters that relax instruction choices to learn programs by gradient descent (Bošnjak et al., 2017; Gaunt et al., 2016), NDVM preserves discrete execution and differentiates the numerical operations on the executed path. Section 3 explains the scope of these gradients and their finite-precision implementation.

The implementation separates program structure from numerical values. One traversal of a program can then evaluate many parameter settings or data points in parallel, with separate gradients for each. This makes the evaluator a reusable inner loop for program-and-parameter co-search: OpenEvolve (Sharma, 2025) proposes a program, NDVM supports fitting its constants, and the resulting behavior determines its score (Section 4). The execution advantage is workload-dependent. We report both short-fit regimes that favor interpretation and a solver workload whose staged execution is faster.

Our main application asks the search to improve a numerical scheme for transporting waves and capturing shocks. A shock is a sharp jump in a solution, such as a pressure front. A useful solver must resolve such features without creating spurious ripples, while remaining accurate where the solution is smooth. Weighted Essentially Non-Oscillatory (WENO) schemes address this problem by combining several local estimates with adaptive weights (Liu et al., 1994; Jiang & Shu, 1996; Shu, 1998). We search for the program that computes those weights, then fit its constants through the full simulation. Thus a proposal is judged by the solution it produces, rather than by how closely its formula resembles a target.

The search recovers the known WENO-Z+ mechanism (Acker et al., 2016) and recalibrates its coefficients. Across six seeds, the resulting schemes reduce error relative to WENO-Z (Borges et al., 2008) by 34–46% on three transfer problems. A fixed seed-0 calibration reduces error relative to published WENO-Z+ by 26–47%. These are bounded improvements: WENO-Z is more accurate on purely smooth data, and a stronger-shock test exposes a small undershoot. The recovered mechanism may be present in the language model's training data; the contribution is its selection and calibration, not an unaided mathematical discovery.

We make three contributions to joint program and parameter search.

- **Reusable differentiable execution.** NDVM accepts supported programs as data and provides reverse-mode derivatives of their numerical outputs, without a separate differentiable implementation for each candidate (Section 3).

- **A practical inner fitting service.** Batched numerical values let one program traversal serve many evaluations. We describe its use inside a language-model search and measure where its execution costs help or hinder fitting (Sections 3–4).

- **A scientific application with explicit controls.** WENO experiments separate mechanism recovery, parameter calibration, and transfer behavior using proposer, prompt, gradient-free, neural, and tuned-family controls (Section 5). A second application exercises state-space programs with loops, branches, and matrices, including failures of structure recovery (Section 6).

2 Related Work

Searching programs and fitting their constants. FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025) use language models to propose code and forward execution to score it. OpenEvolve provides an open implementation of this search pattern (Sharma, 2025). Other systems explicitly fit numerical constants: LLM-SR (Shojaee et al., 2025) and LaSR (Grayeli et al., 2024) propose equation structures for a numerical optimizer, while symbolic regression and sparse dynamics identification fit expressions within an operator grammar or library (Cranmer, 2023; Udrescu & Tegmark, 2020; Brunton et al., 2016). Scientific generative agents (Ma et al., 2024) combine hypothesis proposals with continuous fitting through a differentiable simulator. Our work builds on this separation of discrete search and continuous optimization. Its focus is the reusable execution service beneath that loop: an already-built interpreter accepts new programs as data and differentiates their numerical behavior. In our WENO application, the fitting objective passes through the simulation that uses the proposed program.

Differentiable interpreters and automatic differentiation. TerpreT (Gaunt et al., 2016), differentiable Forth (Bošnjak et al., 2017), and differentiable functional interpreters (Feser et al., 2016) relax program choices to learn discrete structure. Other approaches combine learned components with interpreted code (Gaunt et al., 2017) or learn an interpreter (Macfarlane et al., 2026). Smooth interpretation (Chaudhuri & Solar-Lezama, 2010) and branch-smoothing differentiation (Kreikemeyer & Andelfinger, 2023) modify execution so gradients can cross discrete decisions. NDVM instead keeps program structure and branch decisions discrete. Its gradients describe the numerical computation along a realized execution path; they do not differentiate the choice of that path.

This gradient semantics is shared with automatic differentiation of executed code, including staging (Bradbury et al., 2018), source transformation (Innes, 2018; Moses et al., 2021), and operator-overloading tapes (Hogan, 2014; Carpenter et al., 2015). AD inside language implementations has a substantial history (Pearlmutter & Siskind, 2008). Closest to our interpreter construction, Maleki et al. (2021) add dual numbers to a Scheme interpreter to supply forward-mode derivatives. NDVM uses reverse mode to obtain derivatives with respect to many constants from one scalar objective, and batches numerical values while sharing program traversal. These are implementation and cost distinctions, not a new definition of differentiation. Automatic compiled alternatives also avoid manual rewrites; their preparation and reuse costs must be measured, rather than assumed.

Learning numerical schemes. Learned shock-capturing methods replace or correct WENO weights with neural networks (Stevens & Colonius, 2020; Bezgin et al., 2022); constrained designs such as DSP-WENO (Charles & Ray, 2025) encode desired numerical properties in their architecture. Our neural controls use a generic weight network and an order-preserving design under our training objective. They are not reproductions of the published constrained architectures. The symbolic search instead returns a formula that can be compared directly with the expert WENO lineage (Liu et al., 1994; Jiang & Shu, 1996; Borges et al., 2008; Acker et al., 2016). Section 5 distinguishes recovery of that known mechanism from improvements obtained by calibrating its coefficients, and reports accuracy together with oscillation and convergence.

3 Differentiate the Evaluator, Not the Program

The evaluator is a fixed function $E(P, \theta)$: its inputs are a program P and that program's continuous parameters θ. A downstream computation uses the returned value to form a loss. NDVM supplies the derivatives needed to propagate that loss back to θ. For example, a search can replace $ax + b$ with $a\sin(bx)$ while keeping the same evaluator and optimizer. The formula changes as data; the native executable does not change.

Programs as data. The supported functional language includes arithmetic, transcendental functions, comparisons, conditionals, local bindings, pairs, closures, loops, recursion, and dense matrix operations. The runtime parses and expands a program into a tree, then interprets it. Cached parsing and variable lookups can be reused across parameter updates, but NDVM does not compile each candidate into a specialized native function or computation graph. This differs from partial evaluation,

which specializes an interpreter to a particular program (Futamura, 1971; Jones et al., 1993). The supported numerical and control-flow contract is described in Appendix B.

Separate the structure from the numbers. A program contains discrete objects, such as symbols, pairs, and closures, as well as numerical values. NDVM stores the discrete objects as native records. Scalar numbers occupy dense payload buffers; vectors and matrices have their own numerical storage. A record refers to these values by an index instead of embedding a tensor in every language-level object. The interpreter still performs the program's structural work, but only numerical operations enter the derivative tape. This separation reduces the object creation and tensor dispatch that dominate the eager reference evaluator.

Record execution, then differentiate it. During a forward call, NDVM records each differentiable numerical operation and its operands. A reverse pass visits that tape in the opposite order and accumulates derivatives by the chain rule. Reverse mode obtains gradients with respect to all constants from a scalar loss without a separate forward traversal for each constant. Its cost scales with executed numerical work, including the work introduced by additional parameters.

For the example $P(x) = a\sin(bx)$, the forward traversal records a multiplication, a sine, and another multiplication. Reversing those same operation records gives $\partial P/\partial a = \sin(bx)$ and $\partial P/\partial b = ax\cos(bx)$. The runtime already knows the derivative rule for each operation; it needs no rule written specifically for this formula. Replacing the formula changes the recorded operations, while reusing those rules. If the output enters a simulation, the surrounding differentiable solver propagates the final loss back to each evaluator call. Thus calibration can ask how changing a constant affects the eventual simulated solution, rather than just the weight or intermediate value returned by P.

Throughout the paper, *exact gradients* means automatic differentiation of the executed numerical operations, rather than finite-difference estimates or a smoothed program. It does not mean exact real-number arithmetic. A constant used only in a branch predicate receives no gradient through that predicate; a branch crossing can make the loss nondifferentiable. The historical implementation uses float32 arithmetic. Its two scalar test corpora pass 600 checks over 200 programs and 1,500 checks over 500 programs against an independent eager evaluator and finite differences. These are checks on those corpora, not a proof for every supported program. Appendix B identifies later numerical repairs and separates their source from the runtime used for the historical measurements.

Share a traversal across many evaluations. Each numerical payload can hold B values, called lanes. The lanes share program structure but carry different data or parameter settings. At a branch, active-lane masks determine which result and derivative belong to each lane; incompatible structural results are rejected. This allows one traversal to serve multiple optimizer restarts or many points in a simulation. On the recorded 80-step Kalman-filter benchmark, per-lane runtime with gradients falls from 2.96 ms at $B = 1$ to 0.049 ms at $B = 256$, about $60\times$. Through the host optimizer interface, the measured reduction is $21\times$ (Appendix B).

Where interpretation helps, and where it does not. Avoiding compilation is useful only when its saved preparation cost exceeds the extra cost of interpreting each call. On five recorded programs, from scalar expressions to a matrix recurrence, JAX compilation takes 30 ms to 3.4 s per structure, while NDVM takes 0.15–0.73 ms per warm value-and-gradient call. Under the measured amortization model, staged JAX repays compilation after roughly 213 to 42,000 calls. Reusing a compiled structure lowers that cost; these figures are workload-specific, not a universal advantage over automatic compilation.

WENO illustrates the opposite regime. Here NDVM evaluates a small weight formula inside a much larger host-tensor simulation. Repeated crossings of that interface dominate its cost. Across the 12 parameterized benchmark candidates, median measured times are $1{,}808$ ms per gradient step for NDVM, 421 ms for the eager kernel, and 181 ms for staged JAX, with 4.21 s median JAX compilation. Seven staged rows have failed gradient checks, so their timings do not establish a speedup to equivalent fitting quality. The result nevertheless exposes a costly integration boundary. The service's practical contribution is reusable fitting; execution speed depends on what is interpreted, what remains in the host, and how often a structure is reused.

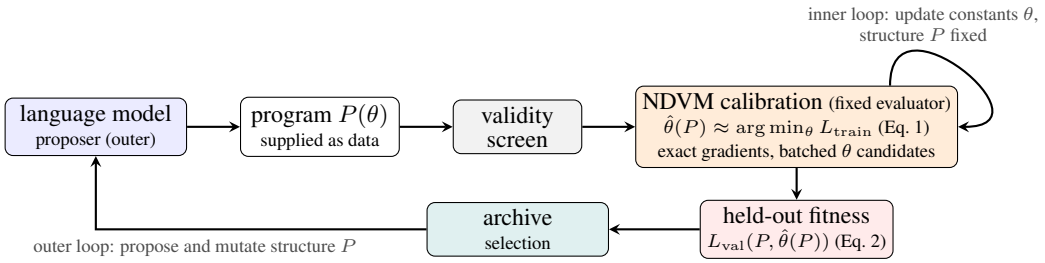


Figure 1: **Program-and-parameter co-search.** A language-model outer loop proposes and mutates program structure P; each survivor is supplied as data to the fixed evaluator, whose exact reverse-mode gradients calibrate its constants θ through the task rollout (inner loop), one structural execution serving the whole batch of parameter candidates; held-out fitness after calibration selects structures into the archive seeding the next round.

4 Program-and-Parameter Co-Search

We now place the evaluator inside a search loop. A candidate has two parts: a program P drawn from the supported language $\mathcal{P}$, and a vector of constants $\theta \in \mathbb{R}^{d(P)}$. The program determines both the computation and the number of constants. Ideally, the inner optimizer would solve

$$\theta^\star(P) = \arg\min_{\theta \in \mathbb{R}^{d(P)}} L_{\text{train}}(P, \theta), \tag{1}$$

and the outer search would select the fitted program with the lowest validation loss,

$$P^\star = \arg\min_{P \in \mathcal{P}} L_{\text{val}}\big(P, \theta^\star(P)\big). \tag{2}$$

In practice we use a finite-budget approximation $\hat{\theta}(P)$, so the score reflects both the program's structure and how successfully its constants were fitted. Training and validation data are disjoint, but validation repeatedly guides program selection; it is not an untouched final test set. Figure 1 shows how the two levels interact.

Outer search. OpenEvolve (Sharma, 2025) uses a language model to mutate programs drawn from an archive of scored candidates, following the evolutionary search pattern of Romera-Paredes et al. (2024); Novikov et al. (2025). The harness screens the extracted numerical program, fits its constants, and evaluates the fitted candidate. Invalid or nonfinite evaluations receive failure scores. The historical harness imports generated Python wrappers before extracting their numerical programs; the grammar screen is therefore not a security sandbox.

Inner calibration. The evaluator binds a candidate's constants to numerical payloads. Adam then takes a fixed budget of steps using derivatives through the task's full rollout. Batched evaluations share a program traversal, while each lane retains its own numerical state. The derivative machinery is shared across structures; a new proposal does not require a new gradient implementation. This provides local gradients, not a guarantee of reaching the global minimum in Equation 1.

Accuracy and constraints. For the WENO task, a low average error is insufficient if the solution develops spurious ripples near a shock. We penalize both error and excess oscillation: $L_{\text{train}} = \text{err} + \lambda[c - \delta]_+$, where $[x]_+ = \max(x, 0)$, c measures oscillation, and δ is its tolerance. Both terms are computed from the same simulated solution. The penalty discourages violations; it does not guarantee feasibility, which we check separately during evaluation.

5 Recovering and Recalibrating a Shock-Capturing Scheme by Co-Search

The first application searches for a small component of a fluid solver: the rule that chooses how to combine neighboring grid values. This is a useful test of co-search because an apparently minor change in a weight formula can alter the accuracy and stability of an entire simulation.

What the solver must do. Smooth solutions can be approximated accurately by fitting a polynomial to nearby grid values. A polynomial that crosses a shock, however, can create artificial oscillations. WENO5 combines three local estimates and reduces the weight of estimates that cross a discontinuity (Liu et al., 1994; Jiang & Shu, 1996; Shu, 1998). Each estimate uses a small set of neighboring points, called a *stencil*. On smooth data the combination should retain fifth-order accuracy: asymptotically, halving the grid spacing reduces spatial error by about $2^5 = 32$. Near a shock it should avoid spurious peaks and troughs. Excess smoothing prevents oscillations but also blurs physically meaningful features; the aim is to balance these effects.

The program being searched. The reconstruction combines three stencil estimates p_k as $\hat{u}_{i+1/2} = \sum_{k=0}^{2} \omega_k p_k$, where $\omega_k = \alpha_k / \sum_j \alpha_j$ and $\alpha_k = h(\beta_k, \tau, d_k)$. Here β_k measures local roughness, τ compares roughness across the wider stencil, and d_k is the fixed weight desired in smooth regions. We search for the function h within the arithmetic grammar of Appendix A. WENO-JS (Jiang & Shu, 1996), WENO-Z (Borges et al., 2008), and WENO-Z+ (Acker et al., 2016) provide successively refined expert baselines. WENO-Z+ includes a reversed smoothness ratio that reduces numerical smoothing, with coefficient $\eta = \Delta x^{2/3}$. The search is not supplied this formula.

Co-search setup. A language-model search (Fable, via OpenEvolve (Sharma, 2025)) proposes h as a small program. NDVM supplies its derivatives inside the advection solver, whose time integration uses SSP Runge–Kutta (Gottlieb et al., 2001). Advection transports a profile without changing its shape, so it supplies an exact target for fitting. The objective combines solution error with an oscillation penalty over smooth and discontinuous initial profiles (Section 4). Appendix D describes solver checks and evaluation conventions; Appendix F reports independent-reference checks.

The discovered scheme. The search converges on a compact closed form,

$$\alpha_k \;=\; d_k\Big(1 \;+\; b^2\,(\tau/(\varepsilon+\beta_k))^2 \;+\; a^2\,(\varepsilon+\beta_k)/(\varepsilon+\tau)\Big), \tag{3}$$

with $a \approx 1.5$ and $b \approx 0.2$ after float64 recalibration (search-time values are in Appendix C). The first correction adjusts the WENO-Z term; the second recovers WENO-Z+'s reversed ratio, replacing its grid-dependent coefficient with a fitted constant. Squared coefficients keep the unnormalized weights positive. The measured smooth-grid convergence orders are $5.20, 5.11, 5.04$. This is recovery and calibration of a known mechanism, rather than proof that the proposer independently derived it.

Transfer to different equations. We test whether a weight rule fitted on advection remains useful elsewhere. The transfer suite contains a new advection profile, Burgers' equation (a nonlinear scalar wave that forms shocks), and the Euler equations at the Sod shock tube (gas expanding across an initial discontinuity). Table 1 retains the original evaluation convention: gains over WENO-Z average 34%, 46%, and 41% across six seeds. All six satisfy the reported Burgers/Euler oscillation threshold. Four recover Equation 3's family and two add damping terms. The seed-0 comparison against published WENO-Z+ gives 25.9–46.6% lower error.

The original Burgers/Euler references use block averages of finer-grid solutions. With independent point references and matched regularization, the primary mean gains are 34.1%, 40.9%, and 32.2%. Improvement persists, but its magnitude depends on the evaluation convention. Furthermore, the Burgers/Euler threshold does not certify oscillation-free advection or stronger shocks (Appendix F).

A stronger-shock test. The Shu–Osher problem sends a shock through a varying-density field (Shu & Osher, 1989). Resolving the resulting fine structure requires low numerical smoothing, but that also risks oscillation. The discovered scheme resolves more post-shock structure while admitting an undershoot of approximately 1.49×10^{-2} that WENO-Z avoids (Figure 2b). It is therefore a different accuracy–robustness tradeoff, rather than an improvement on every metric (Appendix E).

Separating the proposer from calibration. Table 2 crosses two proposers with the presence or absence of gradient calibration. Without gradients, all three Fable runs still recover the reversed-ratio mechanism, but their reported Burgers/Euler gains are smaller: $12.0\%/21.7\%$ versus $45.9\%/40.6\%$. The open-weights proposer returns WENO-Z-family forms in all six runs. Calibration improves some of their errors, but two of three calibrated runs exceed the Euler oscillation threshold. The

Table 1: **A co-searched weight function generalizes across hyperbolic problems and beats WENO-Z.** Original-suite error reduction vs WENO-Z (higher is better); fitted on 1-D advection only, classical baselines at published parameters, the tuned row calibrated identically (Appendix D). "Non-osc.": seeds with Burgers and Euler overshoot below 10^{-3}; $\pm$: std across seeds. Marked rows fail at least one reported criterion († oscillation, reduced order; ‡ Euler overshoot 2.2×10^{-3} in two of three seeds).

Method	Order	Adv. OOD (%)	Burgers (%)	Euler Sod (%)	Non-osc.
WENO-JS (Jiang & Shu, 1996)	5.0	-20.4	-3.7	-12.2	yes
WENO-Z (Borges et al., 2008) (reference)	5.0	0.0	0.0	0.0	yes
WENO-Z+ (Acker et al., 2016)	5.1	$+6.8$	$+0.2$	$+1.6$	yes
Z+ family, tuned by our calibrator	5.1	$+39.6$	$+45.6$	$+38.6$	yes
Co-search (ours), 6 seeds	5.11	$\mathbf{+34.0 \pm 5.4}$	$\mathbf{+45.9 \pm 1.0}$	$\mathbf{+40.6 \pm 2.3}$	6/6
Hint-stripped prompt, 3 seeds	5.10	$+29.5 \pm 1.1$	$+45.4 \pm 0.2$	$+40.3 \pm 0.7$	3/3
Fable only, no exact gradients, 3 seeds	5.11	$+32.1 \pm 1.9$	$+12.0 \pm 2.7$	$+21.7 \pm 3.0$	3/3
Open-weights (Qwen3.6-35B), 3 seeds	5.00	$+11.9 \pm 0.1$	$+56.7 \pm 0.9^\ddagger$	$+49.5 \pm 14.2^\ddagger$	1/3
Open-weights, no exact gradients, 3 seeds	5.00	$+10.6 \pm 0.6$	$+47.5 \pm 3.7$	$+28.8 \pm 1.5$	3/3
Neural weight net (naive), 3 seeds	3.06	-36.9	$+58.4^\dagger$	$+37.0^\dagger$	0/3
Neural weight net (order-preserving), 3 seeds	3.33	-17.4	$+53.7^\dagger$	$+44.3^\dagger$	1/3

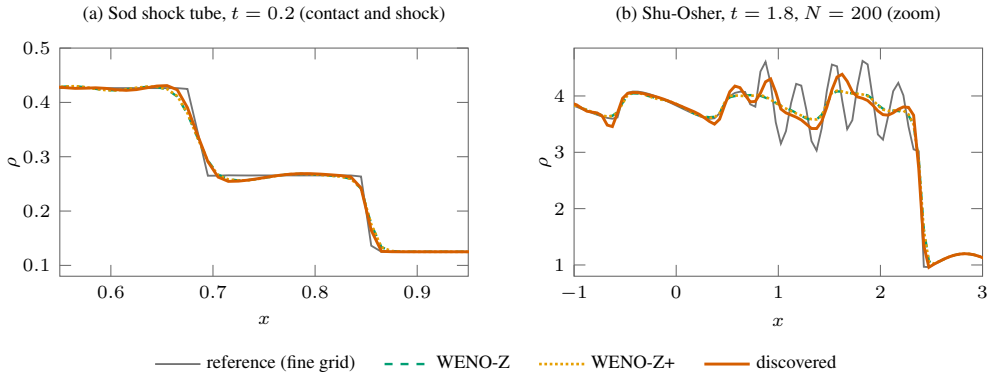


Figure 2: **Density profiles for the transferred scheme** (line styles and color both carry identity). (a) At the Sod contact and shock the discovered scheme, calibrated on advection alone, stays essentially non-oscillatory while sitting closer to the reference than WENO-Z and WENO-Z+ (Table 1; Appendix D). (b) In the amplified post-shock region of the Shu-Osher test it tracks more of the reference structure while admitting the small undershoot discussed in the text: more structure resolved, a thinner robustness margin at the strong shock.

results show that selecting a mechanism, fitting its constants, and satisfying numerical constraints are distinct outcomes. They do not isolate the benefit of calibrating during search from calibrating the final selected family; identical post-selection fitting of the shared family matches its coefficients (Appendix F).

Sensitivity and competing parameterizations. Removing WENO hints from the prompt, or starting from linear weights, still recovers the reversed-ratio mechanism in every recorded seed (Appendix A). This reduces sensitivity to those explicit hints but does not exclude knowledge acquired during model training. When the expert WENO-Z+ family is supplied directly, fitting its coefficients or searching a two-parameter grid matches the discovered scheme's performance (Appendix D). The role of program search is to propose the family and adjustable constants; it is unnecessary once a suitable family is already known.

Why the adjustable structure matters. A stronger expert scheme, WENO-Z+M (Luo & Wu, 2021), makes this distinction concrete. Its reversed-ratio correction contains an exponent: changing that exponent changes how sharply the correction responds to local roughness. Jointly fitting the coefficient and exponent on a smooth training objective moves the exponent from its published value 0.5 to about 1.43. Advection error improves, but the strong-shock range violation grows from about 0.006 to 0.64. A separate exponent sweep shows where that deterioration occurs (Figure 3). The discovered programs keep the exponent fixed at 1.0, removing that optimization direction,

Table 2: **The two-by-two: proposer strength crossed with exact-gradient calibration.** Cells: seeds recovering the β_k/τ mechanism | error reduction vs WENO-Z on out-of-distribution advection / Burgers / Euler (%) | non-oscillatory seeds (‡ Euler overshoot in two seeds; detail in Table 1).

Proposer	Exact-gradient calibration	No gradient calibration
Frontier (Fable)	6/6 | +34.0/+45.9/+40.6 | 6/6	3/3 | +32.1/+12.0/+21.7 | 3/3
Open-weights (Qwen3.6-35B)	0/3 | +11.9/+56.7‡/+49.5‡ | 1/3	0/3 | +10.6/+47.5/+28.8 | 3/3

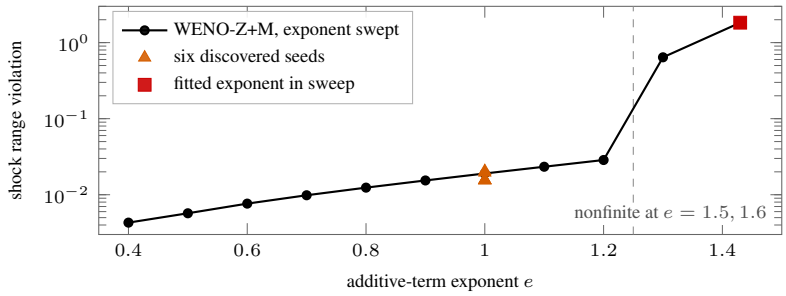


Figure 3: **An exponent can improve fitting while destabilizing shocks.** The WENO-Z+M sweep holds the base coefficient at 579; its shock range violation rises sharply between $e = 1.2$ and 1.3. The square marks the fitted exponent evaluated in this fixed-coefficient sweep, not the jointly fitted scheme. Triangles mark all six discovered programs at their fixed exponent (two pairs overlap). The vertical axis is logarithmic and measures departures above and below the reference density range. All discovered programs retain small nonzero violations. Appendix D gives the settings and both fitting conventions.

although their smaller shock undershoots remain nonzero. Fixing structure can therefore limit a harmful parameter change. The different fitting objectives and residual violations preclude a general robustness guarantee (Appendix D).

A PySR control instead fits pointwise weight-function targets. Its selected expressions transfer worse than WENO-Z, even when its target contains the reversed-ratio mechanism (Appendix I). This tests that pointwise-regression setup, not symbolic regression with an arbitrary simulation-based objective. Our two neural weight controls also expose a limitation of minimizing rollout loss alone: they lose convergence order or exceed oscillation tolerances (Table 1). Together these comparisons motivate evaluating a proposed model by both its fitted accuracy and the properties the scientific task requires.

6 Beyond Algebraic Expressions: State-Space Model Structure

WENO uses a small algebraic weight formula. A second task tests programs that must maintain a hidden state over time: a state-space model describes how that state evolves and produces noisy observations. A Kalman filter scores the resulting predictions. The proposer chooses the state dimension and matrices governing dynamics, observation, and noise (F, H, Q, R); NDVM fits their constants through a fixed T-step filter with loops, branches, and matrix operations. Four synthetic targets have known generating structures, allowing us to compare a good fit with actual structure recovery. We use the same proposer/calibration comparison as WENO, plus random proposals and classical AIC/BIC model-selection controls (Appendix G).

Here *recovery* is stricter than prediction accuracy: the selected program must match the generating structure and place its constants within predefined bands. For example, all three calibrated Fable runs on the trend target predict held-out observations well enough to pass the behavioral criterion, yet none passes full recovery. Finite observations can support a good predictor without uniquely identifying its noise components. Table 3 summarizes full recovery; the appendix reports the separate structural and behavioral checks, including difficult parameter bands retained from the frozen registry.

Structure recovery is mixed. Only the frontier proposer recovers the harder second-order autoregressive form, while an uncalibrated arm recovers the simplest target more reliably. No arm achieves full registered recovery on the target outside the classical model menu. These outcomes distinguish fitting

Table 3: Full state-space recovery out of three seeds per target. A full match requires both structure and parameter bands; a zero does not imply that every fitted predictor is poor. The complete diagnostics and additional controls are in Table 12.

	Exact-gradient calibration		No gradient calibration	
Generating model	Fable	Qwen	Fable	Qwen
Local level	0/3	0/3	0/3	3/3
Linear trend	0/3	0/3	0/3	0/3
Second-order autoregression	1/3	0/3	1/3	0/3
Threshold-dependent noise	0/3	0/3	0/3	0/3

a sequence well from identifying its generating mechanism: repeated selection can favor structurally incorrect models (Section 7).

A separate optimizer diagnostic asks when gradients improve fitting. It uses a direct PyTorch filter to isolate the optimization comparison. As the number of coupled matrix parameters grows, Adam reaches a lower training objective than the tested derivative-free methods (Table 4). This supports gradient-based fitting, but does not measure an NDVM-specific speed or quality advantage.

Table 4: Dense state-space fitting with direct PyTorch gradients: best training objective (lower is better) as parameter count k grows. Budgets count objective evaluations, not matched wall time. The likelihood convention and optimizer settings are in Appendix H.

k	Adam (exact grad.)	Nelder-Mead	Powell	Diff. evolution	Random
2	-67.9	-67.9	-67.9	-67.9	-67.4
4	-72.7	-72.7	-72.7	-67.6	-64.4
6	$\mathbf{-75.4}$	-74.1	-66.8	-43.5	-56.5
10	$\mathbf{-84.1}$	$+39.7$	$+46.1$	-25.0	$+14.8$

7 Limitations

The WENO improvements concern a fixed set of test problems, not arbitrary conservation laws. WENO-Z remains 2.1–2.6$\times$ more accurate on purely smooth data; stronger shocks and out-of-distribution advection expose oscillation (Appendices E and F). A two-dimensional advection check retains fifth-order convergence and an error reduction, but multidimensional systems remain untested. Recovery of a known mechanism is compatible with training-data knowledge, and the tested proposer dependence does not establish that a closed model is necessary in general.

The state-space task demonstrates execution of loops, branches, and matrices, but good likelihood does not guarantee recovery of the generating structure. Repeated validation-based selection, finite data, initialization, and model misspecification can all affect that result. Likewise, exact AD does not guarantee numerical stability or an optimal finite-budget fit. Historical runtime measurements and later numerical repairs are distinguished in Appendix B. The comparisons demonstrate useful capabilities and workload-dependent costs; they do not measure researcher hours, universal speed superiority, or a general advantage over all automatically compiled fitting services.

8 Conclusion

NDVM provides a reusable way to execute changing scientific programs and fit their continuous parameters. In the WENO application, the language model proposes a known numerical mechanism, and calibration improves its transfer performance within the tested regime. The stronger expert WENO-Z+M comparison also illustrates a useful distinction: freeing a continuous exponent can improve fitting while introducing shock oscillation, whereas the recovered discrete forms retain a fixed exponent (Figure 3). A good objective value alone is therefore not enough to select a scientific program. The evaluator makes continuous fitting available across supported structures; the surrounding workflow must still assess numerical validity, transfer behavior, and whether the proposed structure answers the scientific question.

AI USE STATEMENT

Language models act as program proposers in the method itself; their identities, prompts, and settings are documented in Appendix A. Generative AI assistants also helped with method development and mathematical exposition, implementation of the evaluator and experimental code (including synthetic-data generation), experiment design and critique, result analysis and interpretation, literature discovery, and manuscript drafting and editing. They assisted with figure and artifact preparation. AI-assisted implementations were checked using the reported conformance tests, numerical comparisons, and saved experimental records. These checks have the scope stated in the paper; they are not a guarantee of correctness for arbitrary programs. We take responsibility for the final text, claims, references, code, and other artifacts.

ETHICS STATEMENT

The experiments use numerical simulations and synthetic state-space data, without human subjects or personal data. Generated programs require care: the scientific expressions are restricted by a grammar and checked for invalid numerical behavior, but the historical harness first imports the Python wrappers that contain them. Its grammar and timeout checks do not provide a security sandbox. The supplement preserves that behavior for reproducibility and distinguishes archived inspection from executing generated code. The recovered WENO mechanism is prior art, which we cite and compare directly; the paper makes no claim of unaided discovery or universal robustness.

REPRODUCIBILITY STATEMENT

The anonymous supplement contains the sources, fixed input data, saved outputs, available proposal records, and reproduction instructions for this manuscript's experiments. Its index maps tables and figures to their evidence and commands (Appendix J). WENO search-time fitting uses the historical float32 NDVM, whereas the final WENO transfer evaluations use the provided float64 PyTorch evaluator. State-space search and the direct-PyTorch optimizer diagnostic are identified separately. Historical runtime sources and a later numerical-repair variant are both provided, with their distinct scope documented; new repair code does not retroactively change historical measurements. The independent-reference WENO checks in Appendix F include saved solution arrays and analysis code.

Saved-evidence checks and numerical replay require no new language-model requests. Regenerating searches does require a model service and is not expected to reproduce the original proposals bitwise. Checkpoint archives contain all recovered records, including available rejected proposals, but miss candidates evicted between checkpoints and failures before database insertion. The package reports those limits and distinguishes original file hashes from anonymized copies.

REFERENCES

F. Acker, R. B. de R. Borges, and B. Costa. An improved WENO-Z scheme. *Journal of Computational Physics*, 313:726–753, 2016. doi: 10.1016/j.jcp.2016.01.038. The WENO-Z+ scheme.

Deniz A. Bezgin, Steffen J. Schmidt, and Nikolaus A. Adams. WENO3-NN: A maximum-order three-point data-driven weighted essentially non-oscillatory scheme. *Journal of Computational Physics*, 452:110920, 2022. doi: 10.1016/j.jcp.2021.110920.

Rafael Borges, Monique Carmona, Bruno Costa, and Wai Sun Don. An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws. *Journal of Computational Physics*, 227 (6):3191–3211, 2008. doi: 10.1016/j.jcp.2007.11.038. The WENO-Z scheme.

Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable Forth interpreter. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 547–556. PMLR, 2017. URL `https://proceedings.mlr.press/v70/bosnjak17a.html`.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs. `http://github.com/jax-ml/jax`, 2018. Version 0.3.13.

Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016. doi: 10.1073/pnas.1517384113.

Bob Carpenter, Matthew D. Hoffman, Marcus A. Brubaker, Daniel D. Lee, Peter Li, and Michael Betancourt. The Stan math library: Reverse-mode automatic differentiation in C++. *arXiv preprint arXiv:1509.07164*, 2015. doi: 10.48550/arXiv.1509.07164.

Philip Charles and Deep Ray. Learning WENO for entropy stable schemes to solve conservation laws. *SIAM Journal on Scientific Computing*, 47(6):C1196–C1222, 2025. doi: 10.1137/24M1697566. The DSP-WENO scheme; arXiv:2403.14848.

Swarat Chaudhuri and Armando Solar-Lezama. Smooth interpretation. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 279–291. ACM, 2010. doi: 10.1145/1806596.1806629.

Miles Cranmer. Interpretable machine learning for science with PySR and SymbolicRegression.jl. *arXiv preprint arXiv:2305.01582*, 2023. doi: 10.48550/arXiv.2305.01582. URL https://arxiv.org/abs/2305.01582.

John K. Feser, Marc Brockschmidt, Alexander L. Gaunt, and Daniel Tarlow. Differentiable functional program interpreters. *arXiv preprint arXiv:1611.01988*, 2016. doi: 10.48550/arXiv.1611.01988. URL https://arxiv.org/abs/1611.01988.

Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *Machine Learning and Systems (SysML)*, 2018. URL https://cs.stanford.edu/~rfrostig/pubs/jax-mlsys2018.pdf.

Yoshihiko Futamura. Partial evaluation of computation process: An approach to a compiler-compiler. *Systems, Computers, Controls*, 2(5):45–50, 1971.

Alexander L. Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. TerpreT: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*, 2016. doi: 10.48550/arXiv.1608.04428.

Alexander L. Gaunt, Marc Brockschmidt, Nate Kushman, and Daniel Tarlow. Differentiable programs with neural libraries. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1213–1222. PMLR, 2017. URL https://proceedings.mlr.press/v70/gaunt17a.html.

Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43(1):89–112, 2001. doi: 10.1137/S003614450036757X.

Arya Grayeli, Atharva Sehgal, Omar Costilla-Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2409.09359.

Robin J. Hogan. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM Transactions on Mathematical Software (TOMS)*, 40(4):1–16, 2014. doi: 10.1145/2560359.

Michael Innes. Don't unroll adjoint: Differentiating SSA-form programs. *arXiv preprint arXiv:1810.07951*, 2018. doi: 10.48550/arXiv.1810.07951. URL https://arxiv.org/abs/1810.07951. Introduces the Zygote AD tool for Julia.

Guang-Shan Jiang and Chi-Wang Shu. Efficient implementation of weighted ENO schemes. *Journal of Computational Physics*, 126(1):202–228, 1996. doi: 10.1006/jcph.1996.0130.

Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.

Justin N. Kreikemeyer and Philipp Andelfinger. Smoothing methods for automatic differentiation across conditional branches. *IEEE Access*, 11:143190–143211, 2023. doi: 10.1109/ACCESS.2023.3342136. The DiscoGrad tool; arXiv:2310.03585.

Xu-Dong Liu, Stanley Osher, and Tony Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115(1):200–212, 1994. doi: 10.1006/jcph.1994.1187.

Xin Luo and Song-ping Wu. An improved WENO-Z+ scheme for solving hyperbolic conservation laws. *Journal of Computational Physics*, 445:110608, 2021. doi: 10.1016/j.jcp.2021.110608. The WENO-Z+M scheme.

Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B. Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. LLM and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 33940–33962. PMLR, 2024. URL https://proceedings.mlr.press/v235/ma24m.html. The SGA system; arXiv:2405.09783.

Matthew V. Macfarlane, Clément Bonnet, Herke van Hoof, and Levi H. S. Lelis. Gradient-based program synthesis with neurally interpreted languages. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. doi: 10.48550/arXiv.2604.18907. URL https://openreview.net/forum?id=NAORIWBaoO.

Mehrdad Maleki, Barak A. Pearlmutter, and Jeffrey Mark Siskind. Adding Automatic Differentiation to Scheme by differentiating the interpreter. In *Scheme and Functional Programming Workshop (Scheme 2021), co-located with ICFP 2021*, 2021. URL https://icfp21.sigplan.org/details/scheme-2021-papers/7/Adding-AD-to-Scheme-by-Differentiating-the-Interpreter.

William S. Moses, Valentin Churavy, Ludger Paehler, Jan Hückelheim, Sri Hari Krishna Narayanan, Michel Schanen, and Johannes Doerfert. Reverse-mode automatic differentiation and optimization of GPU kernels via Enzyme. In *SC '21: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2021. doi: 10.1145/3458817.3476165.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. doi: 10.48550/arXiv.2506.13131.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pp. 8024–8035, 2019.

Barak A. Pearlmutter and Jeffrey Mark Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):7:1–7:36, 2008. doi: 10.1145/1330017.1330018.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent. GitHub repository, https://github.com/codelion/openevolve, 2025. Open-source implementation of AlphaEvolve; now at https://github.com/algorithmicsuperintelligence/openevolve.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2404.18400.

Chi-Wang Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In Alfio Quarteroni (ed.), *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, volume 1697 of *Lecture Notes in Mathematics*, pp. 325–432. Springer, Berlin, Heidelberg, 1998. doi: 10.1007/BFb0096355.

Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, II. *Journal of Computational Physics*, 83(1):32–78, 1989. doi: 10.1016/0021-9991(89)90222-2.

Ben Stevens and Tim Colonius. Enhancement of shock-capturing methods via machine learning. *Theoretical and Computational Fluid Dynamics*, 34(4):483–496, 2020. doi: 10.1007/s00162-020-00531-1. The WENO-NN scheme.

Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020. doi: 10.1126/sciadv.aay2631.

A CO-SEARCH PROTOCOL

This appendix records the complete protocol behind Sections 4 and 5: the search space and the verbatim proposer prompt, the outer-loop configuration, the per-candidate evaluation pipeline with its validity gates and calibration budget, the exact objective, the budgets of the four co-search arms, and one methodology lesson from an earlier run whose constraint was too loose. Discovered programs per seed are collected in Appendix C; solver validation gates in Appendix D.

Search space and seed program. A candidate is a single expression in the evaluator's language that computes the unnormalized weight α for one stencil from three inputs, $x = \beta_k$ (this stencil's smoothness indicator), $t = \tau$ (the global indicator $|\beta_0 - \beta_2|$), and $w = d_k$ (the optimal linear weight), plus up to four free scalar constants `a`, `b`, `c`, `d`. The harness applies the same expression to all three stencils and normalizes, $\omega_k = \alpha_k / (\sum_j \alpha_j + 10^{-30})$, so only ratios matter. The admitted surface is two-argument arithmetic (`+ - * /`), `abs`, `min`, `max`, `sqrt`, `exp`, `log`, comparisons inside `if`, and `let*` bindings. Every seed starts from the WENO-JS weight (Jiang & Shu, 1996) as the initial program, `(/ w (* (+ 0.000001 x) (+ 0.000001 x)))`. Proposals are diffs against a presented program, OpenEvolve's diff-based mode (Sharma, 2025), so each step edits the expression in place rather than rewriting the candidate file.

Proposer prompt, verbatim. The system message below is given to the proposer in the two gradient-calibrated arms; it is reproduced verbatim, including its formatting. Two properties matter for interpreting Section 5. First, the prompt names WENO-JS and WENO-Z and gives their closed forms as known points to beat; it does not name WENO-Z+ (Acker et al., 2016) or its reversed ratio β_k/τ, though it does urge cleverer use of τ. Second, the runtime's silent-evaluate-to-zero behavior for unknown operators is disclosed, but the static screen below rejects any foreign token before evaluation, so those operators are excluded from admitted expressions. This screen follows import of the generated Python wrapper and is not a security sandbox.

```
You are discovering the NONLINEAR WEIGHT FUNCTION of a 5th-order WENO
    finite-difference scheme for
hyperbolic conservation laws. The model you edit is the SCHEME
    expression inside 'WEIGHT_PROGRAM'
(the unnormalized weight alpha for one stencil). Keep the triple-
    quoted string intact and make ONE
meaningful change per step. A lower held-out score (accuracy under a
    non-oscillation constraint) is
better; the target to beat is WENO-Z.

HARD CONSTRAINTS (a restricted differentiable interpreter runs alpha;
    violating them wastes the eval):
- Output a SINGLE Scheme expression: the UNNORMALIZED weight alpha for
     one WENO5 stencil. It is
  normalized (divided by the sum over the 3 stencils) by the harness,
     so only RATIOS matter.
- Inputs: x = beta (this stencil's smoothness indicator, >= 0; small =
     smooth, large = near a shock),
  t = tau (global indicator |beta0 - beta2|, >= 0; large = a
     discontinuity is nearby),
  w = d (this stencil's optimal linear weight, a positive constant).
     Free scalar params: a,b,c,d.
- alpha MUST be >= 0 for all (beta,tau,d) (else the normalized weights
     are invalid) -- build from
  positive pieces (w, squares, ratios of non-negatives). Numeric
     literals are decimals like 1.0, 0.5.
- Every + - * / takes EXACTLY TWO args. Allowed ops: + - * / abs min
    max sqrt exp log, and (if (COMP
  e1 e2) then else) with COMP in = < > <= >=. Any other head symbol
     silently evaluates to 0.
- You MAY use (let* ((v expr) ...) body).

WHAT MAKES A GOOD WEIGHT (the objective): minimize solution error on a
     MIX of smooth and shock advection
```

```
problems SUBJECT TO staying non-oscillatory (schemes that overshoot at
     shocks are killed). In SMOOTH
regions the weights should return to the optimal linear weights w (so
    alpha ~ w when beta is tiny), for
low dissipation; near SHOCKS (large beta / large tau) the non-smooth
    stencils must be strongly
down-weighted to avoid oscillation. Known points to beat: WENO-JS = w
    /(eps+beta)^2 (ignores tau, over-
dissipative); WENO-Z = w*(1 + (tau/(eps+beta))^2) (uses tau, less
    dissipative -- the target to beat).
Explore alpha shapes that use tau MORE cleverly (higher powers, r-
    adaptive blends, or a smoothness-
gated form) to cut smooth dissipation further while keeping shocks
    clean. ONE meaningful change per
step. A lower held-out score is better.
```

In the no-gradient ablation arm, and only there, the following paragraph is inserted verbatim between the preamble and the hard constraints:

```
IMPORTANT (no-optimizer mode): there is NO parameter optimizer.
    Symbolic params a,b,c,d are FROZEN
to 1.0 and will NOT be tuned. You must choose CONCRETE numeric
    constants yourself (e.g. 2.3, 0.2,
0.05) and refine them across iterations based on the score. Do not
    rely on a,b,c,d as free knobs.
```

Outer loop configuration. All arms use the same evolutionary configuration: 250 outer iterations per seed with 16 candidate evaluations in parallel; an island population (4 islands, population size 150, migration every 25 iterations) whose archive is kept diverse along two feature axes, expression complexity (parenthesis count, capped at 20) and whether the candidate uses τ, with 6 bins per axis; each prompt presents the program to edit together with the three top-scoring and two diverse archive programs. The run seed sets both the search and the archive random state, and checkpoints are written every 25 iterations. There is no cascade evaluation: every proposal that passes the static screen receives the full pipeline below, under a per-candidate timeout of 1200 s (600 s in the forward-only ablation, which takes no gradient steps); evaluation worker processes are recycled after eight candidates.

Per-candidate evaluation pipeline. Each proposal passes through three gates, and any failure returns fitness -10^6 with the rejection reason attached as an artifact that OpenEvolve surfaces in later prompts, so the proposer sees why a candidate failed.

1. *Static screen.* The expression must have balanced parentheses and contain only tokens from the advertised surface (the operators above, the inputs `x`, `t`, `w`, and the constants `a`–`d`); any foreign token is rejected before numerical evaluation of the expression. The historical harness imports the Python wrapper before extracting and screening that expression.
2. *Calibration.* The free constants that actually appear in the expression are calibrated on the training objective (below) by Adam at learning rate 0.03 for up to 20 steps (the evaluator's file default is 30; every campaign job sets 20), each an exact forward-plus-reverse pass through the full solver rollout, with gradient-norm clipping at 1.0 and an early stop on any non-finite score or gradient. Two restarts are run from positive initializations drawn uniformly from $[0.3, 1.3)$ using a fixed generator (seeds 10 and 11), so calibration is deterministic per candidate; within each restart the best-scoring constants along the trajectory are kept (the trajectory may diverge later), and the restart is selected by held-out score at those constants. A candidate with no free constants skips Adam and is scored directly. Candidates that are non-finite under all restarts are rejected.
3. *Positivity.* At the calibrated constants, α must be finite and non-negative (tolerance -10^{-4}) on a 60×60 grid of $(\beta, \tau) \in [0, 3]^2$ for each $d_k \in \{0.1, 0.6, 0.3\}$; otherwise the normalized weights are not a valid convex combination and the candidate is rejected.

The surviving candidate's fitness is the negated held-out objective at the calibrated constants; the outer search maximizes it. Search and calibration run in float32, the runtime's native precision; all final scheme comparisons in Section 5 are re-run in float64 (Appendix D).

The objective, exactly. Let $u_i(P,\theta) \in \mathbb{R}^N$ be the state at time T reached from initial condition u_i^0 by the differentiable WENO5 rollout of candidate P with constants θ, a fifth-order reconstruction advanced by SSP-RK3 time stepping (Gottlieb et al., 2001), and let $u_i^\star$ be the exact reference (advection at unit speed on a periodic grid translates the initial condition rigidly). With $M_i = \max_j u_{ij}^0$ and $m_i = \min_j u_{ij}^0$, the objective over n initial conditions is

$$L(P,\theta) = \frac{1}{nN}\sum_{i=1}^{n}\sum_{j=1}^{N}\left(u_{ij} - u_{ij}^\star\right)^2 \;+\; \lambda\Big[\frac{1}{n}\sum_{i=1}^{n}\mathrm{over}_i - \delta\Big]_+, \tag{4}$$

where $\mathrm{over}_i = \max_j \left[u_{ij} - M_i\right]_+ + \max_j \left[m_i - u_{ij}\right]_+$, with penalty $\lambda = 100$ and strict overshoot tolerance $\delta = 3 \times 10^{-4}$ (the evaluator file's default is 10^{-4}; every campaign job sets 3×10^{-4} through its environment, as the committed job scripts show): mean squared error plus a hinge on the worst overshoot or undershoot beyond the initial condition's range, averaged over the batch. Both terms come from the same rollout, so the calibration gradient sees the constraint as well as the error (Section 4). The grid has $N = 100$ cells on the periodic unit interval at CFL 0.4 and horizon $T = 0.4$, so one rollout is 100 SSP-RK3 steps (a rigid shift of 40 cells); each of the 300 stage evaluations calls the candidate weight program once over 7,200 lanes (3 stencils $\times$ 24 initial conditions $\times$ 100 interfaces) in a single structural walk (Appendix B). L_{train} uses a fixed set of 24 initial conditions (8 smooth random low-wavenumber sine sums; 8 steep Gaussians with widths $\sigma \in [0.02, 0.06]$; 8 square pulses with half-widths in $[0.1, 0.25]$), and L_{val} uses 24 held-out initial conditions drawn from the same generator with an independent seed (generator seeds 0 and 777). Under this objective the WENO-Z anchor (Borges et al., 2008) scores $L_{\mathrm{val}} = 2.03 \times 10^{-3}$, the value every arm must beat.

The four arms. Table 5 summarizes the per-arm budgets and settings; everything not listed is identical across arms, including the evaluator code path of the two gradient arms, the target files, and the objective constants. Each seed of each arm ran on one 48-core CPU node (110 GB memory), evaluations pinned to a single thread (BLAS and torch thread counts set to 1) with 16 candidates evaluated concurrently. Both proposers are driven through the same OpenAI-format chat client with exact model identifiers `claude-fable-5` (frontier arm) and `qwen/qwen3.6-35b` (open-weights arm); requests to Fable omit the sampling fields its endpoint rejects (temperature, top-p, seed), so it samples at endpoint defaults, while the open-weights arm sets temperature 0.9.

Full co-search. Fable proposes structures; the evaluator calibrates constants by exact gradients. Six seeds total: the original strict-campaign discovery (seed 0, one 12-hour job) and five replication seeds run sequentially in one 24-hour job.

Ablation, no gradients. Identical proposer, budget, and objective, but the evaluator is forward-only: a plain vectorized evaluator of the same language scores each candidate directly on the held-out set under the identical hinge (Equation 4), the constants `a`–`d` are frozen at 1.0, the prompt insertion above tells the proposer to pick concrete numeric constants itself, and the positivity gate probes the normalized weights (finite and within $[-10^{-3}, 1+10^{-3}]$ on a 50×50 grid). This compares calibrated and uncalibrated search, but the different positivity screens and final fitting protocols prevent a clean causal attribution to search-time calibration alone.

Hint-stripped prompt. Three further seeds rerun the full arm with the leading content removed from the prompt: no baseline is named, no formula given, and the invitation to use τ more cleverly is dropped; the grammar contract and the objective description remain, since they define the task rather than hint the solution, and the seed program keeps the same starting formula with its name removed (evolution requires a valid starting individual). All three seeds recover the β_k/τ mechanism at holdout objectives inside the full arm's range (1.32 to 1.36×10^{-3}), and the float64 transfer evaluation reproduces the full arm on the two shock problems while sitting at the bottom edge of its advection seed range: $+29.5 \pm 1.1$ (out-of-distribution advection, full-arm seed minimum $+29.6$), $+45.4 \pm 0.2$ (Burgers), $+40.3 \pm 0.7$ (Euler), three of three admissible, measured order 5.10. The runs completed in 51 minutes total.

Open-weights proposer. The proposer is swapped for qwen3.6-35b, an open-weights model served through an OpenAI-compatible endpoint with thinking mode enabled and a 32,768-token completion budget; the model's reasoning trace (delivered either inline or in a separate response field, depending on server configuration) is stripped before diff parsing so the outer loop sees only the final answer. Requests use a 900 s timeout with 5 retries at 10 s spacing to absorb transient endpoint errors. Evaluator, objective, calibration budget, and outer budget are identical to the full co-search; three seeds were run, with a pre-commitment to report the outcome either way.

Open-weights ablation. The fourth arm crosses the two changes: the open-weights proposer of the arm above drives the forward-only, frozen-constant evaluator of the ablation arm, under the same budget and objective (three seeds, run exactly as the ablation arm with the proposer swapped). Together the four arms form the two-by-two of Section 5. Per-seed outcomes for the co-search arms, and the neural baselines' aggregates, are in Appendix C.

Table 5: Budgets and settings per co-search arm. "Adam $\leq 20 \times 2$" abbreviates up to 20 exact-gradient steps at learning rate 0.03, two restarts. Wall-clock limits are Slurm job limits: the full arm ran seed 0 in one 12 h job and seeds 1–5 sequentially in a 24 h job; the other arms ran their three seeds sequentially in one job. Shared rows are repeated to make the controls explicit.

	Full co-search	Ablation (no gradients)	Open-weights proposer
Proposer	Fable	Fable	qwen3.6-35b, thinking
Completion budget (tokens)	8,192	8,192	32,768
Sampling temperature	endpoint default	endpoint default	0.9
Request timeout / retries	600 s / client default	600 s / client default	900 s / 5
Independent seeds	6	3	3
Outer iterations $\times$ workers	250×16	250×16	250×16
Islands / population / migration	4 / 150 / 25	4 / 150 / 25	4 / 150 / 25
Inner calibration	Adam $\leq 20 \times 2$	none (frozen at 1.0)	Adam $\leq 20 \times 2$
Per-candidate timeout	1200 s	600 s	1200 s
Tolerance δ / penalty λ	3×10^{-4} / 100	3×10^{-4} / 100	3×10^{-4} / 100
Wall-clock limit	$12 + 24$ h	12 h	60 h

Measured stream accounting. The wall-clock rows above are Slurm allocations; the archived program databases (checkpoints every 25 iterations, shipped in the supplement) measure the runs themselves. Per 250-iteration full-arm run: 216 to 239 distinct evaluated proposals are recoverable, 132 to 145 scoring and 73 to 97 rejected (the rejections are dominated by the static grammar screen, 447 across the six runs, plus 74 calibration-instability kills), with first-to-last-candidate wall-clock of 19 to 23 minutes and a median inter-candidate gap of 2.4 to 3.9 s at 16 concurrent evaluations. The archive preserves candidate code and scores, with prompt and completion fields where recorded, and a supplement script replays the archived trajectories through the deterministic evaluator without re-sampling the language model.

The lenient-tolerance trap. An earlier campaign ran this loop with the overshoot tolerance set to $\delta = 2 \times 10^{-3}$. The search did what it was asked: it returned a scheme that beat WENO-Z on that objective. Under stricter post-hoc testing the scheme collapsed: on a hard square-wave advection test its overshoot reached 8×10^{-3}, more than an order of magnitude above the 5.5×10^{-4} that even the dissipative WENO-JS admits on the same test, so its accuracy had been bought by weak limiting, exactly the trade a usable WENO scheme must not make. The protocol above is the repaired version: the tolerance was tightened to $\delta = 3 \times 10^{-4}$, at the overshoot level the classical schemes themselves attain on the training mix, and the horizon was chosen long enough ($T = 0.4$, 100 steps) that slowly growing oscillation registers in the training objective itself, since short rollouts hide it. Every number in Section 5 comes from the strict campaign. We record the lenient run as a methodology lesson rather than a machinery failure: an effective co-search optimizes exactly the objective it is given, including any slack a constraint leaves, so constrained discovery requires setting the constraint at the level actually required and validating out of regime before any claim; the independent cross-regime gates of Appendix D exist for that reason.

Tolerance and horizon sensitivity. With the six discovered structures frozen and only their constants recalibrated per cell, a sweep over $\delta \in \{10^{-3}, 5 \times 10^{-4}, 3 \times 10^{-4}, 10^{-4}\}$ and training horizons $T \in \{0.2, 0.4, 0.8\}$ (evaluation protocol fixed strict) finds every one of the twelve cells six of six

non-oscillatory, with mean transfer gains moving smoothly between $+31.4$ and $+36.6\%$ (out-of-distribution advection), $+39.1$ and $+48.9\%$ (Burgers), and $+38.4$ and $+40.9\%$ (Euler): tighter tolerances and longer horizons trade a little Burgers gain for a little out-of-distribution gain, the dissipation knob behaving as designed, with no cliff anywhere in the range. The campaign cell ($\delta = 3 \times 10^{-4}$, $T = 0.4$) reproduces the published means exactly. This sweep measures calibration and evaluation sensitivity at fixed structure; the search-level sensitivity evidence is the lenient-trap contrast above. Per-cell, per-seed values ship with the supplement.

B THE EVALUATOR IN DETAIL

This appendix expands Section 3: the value representation, the reverse-mode tape and its gradient semantics, the measured cost anatomy of the eager alternative that motivated the design, the batching and lane-masking mechanics, the differential validation, the staging comparison behind the amortization claims, and the mechanics by which the evaluator hosts the weight program of Section 5. The reported timings are historical measurements on one CPU core of an HPC compute node (PyTorch 2.12 for the tensor-hosted baselines). Forward and gradient checks have distinct scopes; in particular, the workload benchmark below retains seven flagged staged-gradient results.

Value representation. Every runtime value is a compact scalar record $\langle tag, aux, pid \rangle$: tag is a small integer type discriminator; aux holds an immediate (an interned symbol id, a heap address, a closure id, or a small integer); and pid indexes the numeric payload table when the value is numeric, and is otherwise a sentinel. No tensor or heap object is allocated for a non-numeric value. Scalar payloads live in dense primal and adjoint buffers. Vector and matrix cells have their own shape metadata and numerical arrays, addressed through the structural record. Both storage paths carry the batch axis B, so one structural value can refer to B numerical evaluations. The consequence is the representation change the design rests on: instead of carrying B copies of a tagged value for every runtime object, the evaluator carries one scalar structural object plus B numeric elements, and only when the value is numeric. Tags, symbols, and addresses stay scalar; only numbers pay for the batch.

Heap, environments, and symbols. The language subset excludes mutation, so the heap is a write-once arena: pair construction appends a cell and returns a value whose aux is the cell address, and the pair accessors are direct arena loads. Closures store a code entry and a captured environment; vectors store a slice into an element arena. All symbols are interned to integers, so no string comparison occurs on a hot path, and environment lookups use inline caches keyed by environment shape and symbol so that repeated lookups skip the search. None of these operations propagate gradients; they shape the realized trace. The native evaluator walks the parsed program tree, dispatches its forms, and implements lazy conditionals, variable-length loops, and trampolined tail calls. The object program remains data throughout; it is not compiled into a candidate-specific instruction stream. Structural caches (a decoded-form cache and inline lexical-address caches) and allocation pooling (frame and argument-vector pools) memoize repeated work without compiling any program into its own graph; they cut the matrix-rollout evaluation about $3.2\times$ from the un-optimized native runtime and roughly halve small allocation-bound programs.

The tape, and what its gradients mean. Differentiation is define-by-run reverse mode recorded as a compact native tape over payloads only. The forward pass appends a tape node for each differentiable numeric primitive executed (add, multiply, exp, log, matmul, reduce, and so on); structural operations, symbol lookup, tag tests, pair allocation, closure construction, branch dispatch, emit nothing. The backward pass replays the tape in reverse, accumulating into the dense adjoint buffers with vectorized loops over the batch axis, so tape length tracks the numeric work actually executed, not program size or the evaluator's own recursion depth. The intended semantics, stated as such rather than as a proved theorem, is trace-constant scoping: for inputs that do not change which branches are taken, the output is a composition of differentiable numeric primitives over the realized trace and replay computes its exact gradient; a parameter that appears only in a branch predicate receives no gradient through that predicate; at inputs where a branch flips, the output inherits the source program's own nondifferentiability. These are the semantics eager reverse-mode autodiff over an interpreter already provides; the contribution is making them cheap, and nothing is smoothed to obtain them. Floating-point rounding, primitive-domain conventions, and branch crossings still limit what a numerical gradient check can establish.

Cost anatomy of the eager alternative. The design responds to a measured diagnosis of the eager route, an implementation of the same evaluator in a tensor framework (Paszke et al., 2019) in which every runtime value is a tagged tensor (a one-hot type tag concatenated with a numeric payload). Profiled over five programs spanning regimes, two scalar closed forms, a transcendental expression, a 16-step recursive loop, and an 80-step 2×2 Kalman-filter negative log-likelihood with per-step inverse, determinant, and matrix products, the decomposition is stable: value boxing accounts for 49–61% of forward time and evaluator walking for a further 25–40%, together 85–90%, and this combined share barely moves across a roughly $2300\times$ range of absolute cost, from a 3.1 ms scalar expression to a 7.0 s Kalman forward. Raw arithmetic, including the matrix work inside the Kalman filter, is about 1% or less. In call counts, one Kalman forward issues about two million (1,994,674) value-boxing calls against 35,095 evaluator-walk calls and 3,442 arithmetic calls; counted directly, it allocates 273,238 tagged tensors, about 1.9 MB of peak transient memory. The split representation carries the identical trace in 972 dense payload slots, a $281\times$ reduction in representational object traffic; across the suite the reduction is 26 to $281\times$, growing with program size. The backward pass is 0.4–6% of total eager time, so the forward walk, not gradient propagation, was the target. One further baseline property motivates batching: raising B from 1 to 1024 raises eager forward time by under about 4%, a marginal cost of roughly $0.1\ \mu s$ per additional lane, because the multi-millisecond bulk of each forward pass is per-walk overhead paid once regardless of B. Nor is the gap an artifact of a weak eager encoding: a tuned eager variant that strips the tagged boxing recovers roughly four to five times of it, leaving an 8 to $2133\times$ residual across the suite that the walk itself imposes.

Batching and divergence mechanics. The native runtime turns that batch independence into throughput. The structural walk, the heap, the environments, and the tape are scalar and shared; only the payload buffers are wide, so a single walk serves a population of parameter vectors, restarts, cells, or data points and the backward pass returns per-lane gradients. On the 80-step Kalman likelihood with gradients, per-lane cost falls from 2.96 ms at $B{=}1$ (an uncached walk) to 0.049 ms at $B{=}256$, about $60\times$; measured end to end through the host optimizer boundary, where the runtime is exposed as a differentiable tensor op, the deployed amortization is about $21\times$ (0.63 to 0.029 ms), because caching and tensor marshaling at the boundary shrink the per-walk overhead the batch amortizes. When a branch test diverges across the population, the runtime evaluates each side under its active lane subset and merges per lane, with adjoints gated to the lanes that took each path, so a terminated lane's stale value cannot corrupt another lane's gradient; this is validated by decomposing a batched divergent run into independent single-lane runs and checking that forward outputs and per-lane gradients agree. The amortization therefore suits populations that largely share a control path. Orthogonally, independent candidate evaluations own all of their mutable state and share nothing, so a thread pool with one interpreter per worker scales near-linearly across cores, about $15\times$ on 16 cores of a 32-core node, with results byte-identical to a serial run.

Differential validation. Beyond hand-written conformance cases, the runtime is validated by randomized differential testing against the independent eager tagged-tensor implementation of the same evaluator. A typed generator emits well-typed, scalar-valued, parameter-dependent programs over the scalar control-and-arithmetic fragment of the language: numeric literals, bound parameters and locals, the binary operators $+\ -\ *\ /$, the unary operators $\sin, \cos, \exp, \sqrt{\cdot}, \log, |\cdot|$, `let`/`let*` bindings, `if` with comparison guards, and bounded loops with compile-time-constant trip counts; domains are wrapped so that a NaN or infinity never defeats a gate, and the corpus is frozen to disk for reproducibility. Each program passes three gates: G1 compares the forward value against the eager oracle (float32 tolerance, $\mathrm{atol} = \mathrm{rtol} = 10^{-4}$); G2 compares the per-parameter reverse-mode gradient against the oracle's autograd gradient ($\mathrm{atol} = \mathrm{rtol} = 2 \times 10^{-3}$); G3 compares the gradient against a central finite difference of the runtime's own forward ($\mathrm{atol} = 10^{-2}$, $\mathrm{rtol} = 5 \times 10^{-2}$). The canonical corpus of 200 programs passes all 600 checks across every feature family, and an independent 500-program corpus passes all 1500. The scope caveat is explicit: randomized generation covers the scalar fragment only; the matrix primitives (product, determinant, log-determinant, inverse) are exercised instead by the hand-written 80-step Kalman conformance case, whose noise-covariance gradients through the matrix-adjoint path match the oracle to about 3×10^{-6}, and randomized generation over matrix-shaped, list, and closure programs remains future work.

Staging and eager baselines, quantified. Table 6 details the comparison summarized in Section 3. For each of the five programs, the identical math was hand-written in JAX (Bradbury et al., 2018;

Frostig et al., 2018) and staged into a compiled forward-plus-gradient graph; all staged forward values match the eager oracle to float32. Once compiled, staged execution is fast, 0.007 to 0.016 ms per step for the scalar and loop programs and 0.649 ms for the Kalman rollout, but compilation costs 30.3 to 54.5 ms per fresh structure for the scalar and loop programs and 3412 ms for the Kalman program. The evaluator consumes the same programs as data with no staging at 0.150 to 0.730 ms per forward-plus-gradient step (runtime caches warm). Solving $\text{stage} + n\,t_{\text{staged}} < n\,t_{\text{unstaged}}$ per program, staging amortizes only after $n^\star \in [213,\ 4.2 \times 10^4]$ gradient steps per candidate (a `sympy.lambdify` staging of the closed-form scalar programs after 45 to about 2900); against the eager tagged-tensor evaluator it amortizes almost immediately ($n^\star \in [0.33,\ 10.6]$). The inner-loop budgets of Appendix A, tens of gradient steps per candidate, sit well below the crossover. A staging system can do better when structure repeats: a compilation cache keyed on program structure pays the compile once per distinct skeleton and reuses it across candidates differing only in constants. Measured on six closed-form skeleton families instantiated at r constant-settings each (all conditions agreeing to float32), batched lanes are about $4\times$ faster than structure-cached staging at $r{=}1$, where the 70 to 96 ms per-skeleton compile is unamortized, and cached staging overtakes at about 80 reuses per skeleton (the runtime leads through $r{=}64$ and is overtaken by $r{=}128$). Two caveats carry over from that measurement: the crossover is specific to the small per-candidate numeric work of these programs, and heavier per-candidate budgets, more gradient steps or larger numeric work per candidate, lower it toward staging. In that benchmark, staging benefits from repeated reuse of the same skeleton. The WENO search instead uses 20 steps and two restarts per proposed expression. Its archived streams allow us to measure structural repetition: canonicalizing every recovered candidate by the same skeleton convention as the crossover analysis, the six full-arm streams show novelty 0.61 to 0.68, a mean of 1.5 to 1.6 proposals per distinct skeleton (open-weights streams 2.5 to 3.2). One scoping caution: these suite-derived crossovers govern program-dominated workloads; on the WENO workload itself the whole-rollout measurement below applies, and skeleton caching would change little in either direction there.

The workload benchmark. The original WENO benchmark evaluates 16 saved skeletons, of which 12 have fitted parameters. Restricting all three strategies to those 12, median per-step times are $1{,}808$ ms for NDVM, 421 ms for the eager kernel, and 181 ms for JAX. Median recorded end-to-end times at the 40-step budget are 74.5, 17.4, and 10.5 s, respectively. JAX compilation has median 4.21 s in that same subset; the smaller 2.9 s figure obtained from a different denominator must not be paired with these per-parameterized-candidate summaries. NDVM crosses its host boundary at all 300 stage evaluations of the rollout, explaining why its low-cost numerical kernel does not translate into low full-solver cost.

Seven of the 12 staged rows are explicitly marked provisional after failing gradient checks, including the discovered-family candidates. Their timings are preserved as measurements of those executions, not as times to valid equivalent fits. The records do not establish an inherent JAX limitation or a universal remedy. Similarly, concurrent search throughput and model latency do not by themselves isolate the bottleneck of a sequential candidate fit. Peak resident-set increase for the NDVM strategy was 540 MB, including extension loading.

Runtime versions and later numerical repairs. The source package distinguishes the float32 implementation used for these historical searches and timings from a later repaired runtime. The latter improves reverse accumulation precision, prevents unused nonfinite operations from contaminating live gradients, and fixes scalar broadcasting and final-binding macro expansion. It also supplies explicit float64 execution. The historical scalar test results above remain records of their stated corpora; they do not rule out these subsequently identified defects. Repaired sources and regression tests are included separately. The historical scientific searches were not rerun for this submission, and no old timing is attributed to the repaired implementation. The WENO float64 post-selection evaluation uses its separate PyTorch expression evaluator, not the old native reverse pass.

Hosting the WENO weight program. In Section 5 the candidate object is a program mapping one stencil's inputs to an unnormalized weight: the smoothness indicator β_k, the global reference τ, and the optimal linear weight d_k enter as inputs, and the program's free constants enter as parameters. The lane mapping does the amortization: each reconstruction issues one evaluator call whose lane vector stacks the three stencils across every cell interface and every initial condition in the training batch, $3NM$ lanes for N interfaces and M initial conditions, with constants broadcast to all lanes as

Table 6: Per-program cost of the three evaluation routes for a runtime-proposed program: the eager tagged-tensor evaluator, the native evaluator (NDVM), and hand-written staged JAX. Entries are median milliseconds on one CPU core at B=1; eager and NDVM columns are per-step forward-plus-gradient with no staging, JAX columns are the compiled per-step call and the one-time staging cost per fresh structure. All forward values agree with the eager oracle to float32. The crossover $n^\star$ is the number of gradient steps per candidate after which staging amortizes; it is computed as staging cost divided by the difference in per-step costs, using the unrounded values in the archived JSON.

	per-step fwd+grad		staged JAX		crossover $n^\star$	
program	eager	NDVM	fwd+grad	staging	vs NDVM	vs eager
scalar expression	3.29	0.159	0.007	34.7	229	10.6
Michaelis–Menten	4.72	0.157	0.014	30.3	213	6.4
damped oscillator	10.61	0.171	0.016	38.4	249	3.6
logistic-map loop (16 steps)	163.07	0.150	0.014	54.5	403	0.33
Kalman rollout (T=80)	7133.55	0.730	0.649	3412.1	42,185	0.48

payload entries, so one structural walk of the proposed program prices the entire grid. The returned unnormalized weights are reshaped to the three stencils and normalized to a convex combination in the host tensor code; because the runtime is exposed to the host as a differentiable tensor op, its exact per-lane gradients compose with the host's autograd through the normalization, the fifth-order reconstruction, and the SSP Runge-Kutta rollout (Gottlieb et al., 2001), so the training gradient of the objective with respect to the program's constants is exact along the realized trace of the full solve. The integration is gated before any search: with the hosted program set to a parameterized WENO-Z form, written as data, `(* w (+ 1.0 (* a (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))))))`, which recovers WENO-Z exactly at a=1, the evaluator-computed weights and the full advection solve match an independent tensor implementation of WENO-Z, and the gradient of the solution error with respect to a at a=1.4 matches a central finite difference within the gate's 5×10^{-2} relative tolerance. The gradient gate deliberately uses a steep profile: on smooth data the nonlinear correction is inactive and the gradient with respect to its coefficient degenerates toward zero, so a smooth-profile check would pass vacuously.

C Discovered programs across seeds and arms

This appendix reproduces, verbatim, the winning weight program of every co-search run reported in Section 5: six runs of the full arm (frontier proposer plus exact-gradient calibration), three runs of the no-gradient ablation arm, three runs of the open-weights proposer arm, and three runs of the open-weights ablation arm (open-weights proposer, no gradient calibration). All fifteen searches start from the identical seed program, the WENO-JS weight $\alpha_k = d_k/(\varepsilon + \beta_k)^2$ (Jiang & Shu, 1996), written as `(/ w (* (+ 0.000001 x)(+ 0.000001 x)))`, and run under the same objective, strict non-oscillation constraint, and outer budget of 250 iterations (Appendix A). The proposer writes one expression for the unnormalized α_k of a single stencil over the variables $x = \beta_k$ (that stencil's smoothness indicator), $t = \tau = |\beta_0 - \beta_2|$ (the global smoothness reference), and $w = d_k$ (the optimal linear weight), with free scalar parameters `a,b,c,d`; the harness normalizes $\omega_k = \alpha_k / \sum_j \alpha_j$, so only ratios matter. The literal `0.000001` is the regularizer $\varepsilon = 10^{-6}$ of Equation 3. Each listing below quotes the `WEIGHT_PROGRAM` string from `best/best_program.py` in the named output directory, unedited. For compact canonical forms we write $r_k = \tau/(\varepsilon+\beta_k)$ and $s_k = (\varepsilon+\beta_k)/(\varepsilon+\tau)$; note that no winning program in any arm uses a conditional or a transcendental primitive, so every winner is a rational function of (β_k, τ, d_k).

Full arm: frontier proposer with exact-gradient calibration, six seeds. The original run (`out_weno_strict`) and five independent replications (`out_weno_s1` to `out_weno_s5`):

```
;; original run (out_weno_strict)
(* w (+ (+ 1.0 (* (* b b) (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))
    ))) (* (* a a) (/ (+ 0.000001 x) (+ 0.000001 t)))))

;; replication seed 1 (out_weno_s1)
(* w (+ (+ 1.0 (* (* c c) (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))
    )))
```

```
      (* (* a a) (/ (+ 0.000001 x) (+ 0.000001 t)))))

;; replication seed 2 (out_weno_s2)
(* w (+ 1.0 (+ (* (* a a) (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))
    )) (* (* b b) (/ (+ 0.000001 x) (+ 0.000001 t))))))

;; replication seed 3 (out_weno_s3)
(* w (+ 1.0 (+ (* (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))) (/ (+
    1.0 (* (* c c) (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))))) (+
    1.0 (* (* b b) (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x)))))))
    (/ (* (* a a) (/ (+ 0.000001 x) (+ 0.000001 t))) (+ 1.0 (* (* d d)
     (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x)))))))))

;; replication seed 4 (out_weno_s4)
(* w (+ 1.0 (+ (* (* b b) (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))
    )) (* (* a a) (/ (/ (+ 0.000001 x) (+ 0.000001 t)) (+ 1.0 (* (/ t
    (+ 0.000001 x)) (/ t (+ 0.000001 x)))))))))

;; replication seed 5 (out_weno_s5)
(* w (+ 1.0 (+ (* (* a a) (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))
    )) (* (* c c) (/ (+ x 0.000001) (+ t 0.000001))))))
```

Four of the six (the original and seeds 1, 2, 5) realize exactly the form of Equation 3, $\alpha_k = d_k\,(1 + p^2 r_k^2 + q^2 s_k)$, differing only in which parameter names play the two roles (Z-term damping p, β_k/τ coefficient q): $(p, q) = (b, a)$, (c, a), (a, b), and (a, c) respectively, plus immaterial permutations of operand order in the ε sums and of the association of the three summands. The remaining two are richer members of the same family that keep both corrections. Seed 3 is a rational elaboration, $\alpha_k = d_k\big(1 + r_k^2\,(1 + c^2 r_k^2)/(1 + b^2 r_k^2) + a^2 s_k/(1 + d^2 r_k^2)\big)$, gating both terms by powers of r_k; it was the best of the six on out-of-distribution advection. Seed 4 damps the β_k/τ correction where a discontinuity is flagged, $\alpha_k = d_k\big(1 + b^2 r_k^2 + a^2 s_k/(1 + r_k^2)\big)$; it was the best of the six on Euler. The squares on all free parameters, which keep α_k positive, were proposed in every seed. For the original run the search-time calibration returned $a = 1.317$, $b = 0.194$, that is, a β_k/τ coefficient $a^2 \approx 1.73$ and a Z-term damping $b^2 \approx 0.038$; before any transfer comparison the evaluation harness re-parses each seed's program and recalibrates its constants in float64 through the same advection objective, landing in the same regime ($a^2 \approx 2.3$, $b^2 \approx 0.045$, the $a \approx 1.5$, $b \approx 0.2$ quoted with Equation 3). Table 7 gives the per-seed spread behind the mean and standard deviation of Table 1; the advection maximum belongs to the seed-3 variant and the Euler maximum to the seed-4 variant.

Table 7: Full arm, spread across the six independent searches (original plus five replications): improvement over WENO-Z on the float64 evaluation suite. Every seed beats WENO-Z on all three problems; every seed's measured smooth convergence order is 5.11 and its Burgers and Euler overshoot stays below 10^{-3}.

Improvement over WENO-Z	Mean $\pm$ std	Range	Seeds beating WENO-Z
Advection (OOD)	$+34.0 \pm 5.4\%$	$+29.6\%$ to $+44.1\%$	$6/6$
Burgers (post-shock)	$+45.9 \pm 1.0\%$	$+44.9\%$ to $+47.1\%$	$6/6$
Euler (Sod)	$+40.6 \pm 2.3\%$	$+38.9\%$ to $+44.7\%$	$6/6$

Ablation arm: same proposer and budget, no gradient calibration, three seeds. In this arm the evaluator scores each program forward only; the free parameters are bound to 1.0, so the proposer must write concrete numeric constants itself (`out_weno_abl_s1` to `out_weno_abl_s3`):

```
;; ablation seed 1 (out_weno_abl_s1)
(* w (+ (+ 1.0 (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x)))) (* 6.0
    (/ (+ x 0.000001) (+ t 0.000001)))))

;; ablation seed 2 (out_weno_abl_s2)
(* w (+ (+ 1.0 (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x)))) (* 3.0
    (/ (+ 0.000001 x) (+ 0.000001 t)))))

;; ablation seed 3 (out_weno_abl_s3)
```

```
(* w (+ 1.0 (+ (* (/ t (+ 0.000001 x)) (/ t (+ 0.000001 x))) (* 6.0 (/
     (+ 0.000001 x) (+ 0.000001 t))))))
```

All three are the same structure, $\alpha_k = d_k\,(1 + r_k^2 + C\,s_k)$: the β_k/τ mechanism is recovered in every seed with no gradients, confirming that structure selection does not depend on the inner loop. What differs is the constants. The Z term is left undamped at the frozen coefficient 1 and the proposer's self-chosen β_k/τ coefficients are the round numbers $C = 6$, 3, and 6, where exact-gradient calibration selects approximately 2.3 together with a Z-term damping of approximately 0.045, a far less dissipative operating point. The evaluation harness scores these programs exactly as written (no float64 recalibration, matching the arm's premise). Held-out objective: 0.00149 to 0.00159 across the three seeds, versus 0.00131 to 0.00137 for the full arm and 0.00203 for WENO-Z; transfer stands at $+32.1\%$ (out-of-distribution advection), $+12.0\%$ (Burgers), and $+21.7\%$ (Euler), the ablated row of Table 1. The constants, not the structure, are what separate the two arms on the transfer problems.

Open-weights proposer arm: qwen3.6-35b, identical inner loop and budget, three seeds. This arm swaps only the proposer, replacing the frontier model with the open-weights `qwen/qwen3.6-35b` (thinking mode, 32K completion budget) while keeping the identical exact-gradient inner loop, objective, constraint, and budget (`out_weno_qw_s1` to `out_weno_qw_s3`):

```
;; open-weights seed 1 (out_weno_qw_s1)
(* w (+ 1.0 (* (/ (* a t) (+ x 0.000001)) (/ (* a t) (+ x 0.000001))))
   )

;; open-weights seed 2 (out_weno_qw_s2)
(* w (+ 1.0 (/ (* a t) (+ 0.000001 x))))

;; open-weights seed 3 (out_weno_qw_s3)
(* w (+ 1.0 (* a (/ t (+ 0.000001 x)))))
```

All three winners are WENO-Z-family forms (Borges et al., 2008) carrying a single calibrated coefficient: seed 1 the power-2 form $\alpha_k = d_k\,(1 + a^2 r_k^2)$, and seeds 2 and 3 the identical power-1 form $\alpha_k = d_k\,(1 + a\,r_k)$. None of the three contains the β_k/τ term; at this budget the open-weights proposer never proposed the mechanism that every frontier-proposer search selected. Under the same held-out objective the calibrated winners score 1.779×10^{-3} (best found at outer iteration 177), 1.786×10^{-3} (iteration 97), and 1.786×10^{-3} (iteration 75), each with one free parameter: better than WENO-Z (2.03×10^{-3}) because the inner loop still calibrates the coefficient exactly, but short of both frontier-proposer arms (full arm 1.31 to 1.37×10^{-3}; even the no-gradient ablation reaches 1.49 to 1.59×10^{-3}). Its float64 transfer evaluation (identical protocol) gives out-of-distribution advection $+11.9 \pm 0.1\%$, Burgers $+56.7 \pm 0.9\%$, and Euler $+49.5 \pm 14.2\%$ over WENO-Z at measured order 5.00, but only one of three seeds stays within the non-oscillation tolerance (the other two reach Euler overshoot 2.2×10^{-3}): with no richer mechanism to exploit, the calibration shrinks the τ correction toward the linear weights and spends the stability margin.

Open-weights proposer without calibration: completing the two-by-two. The final arm crosses the open-weights proposer with the forward-only evaluator (constants self-picked and frozen, `out_weno_qwabl_s1` to `out_weno_qwabl_s3`):

```
;; open-weights, no gradients, seed 1 (out_weno_qwabl_s1)
(* w (+ 1.0 (/ (* t 0.15) (+ 0.000001 x))))

;; open-weights, no gradients, seed 2 (out_weno_qwabl_s2)
(* w (+ 1.0 (/ (* 0.01 (* t t)) (* (+ 0.000001 x) (+ 0.000001 x)))))

;; open-weights, no gradients, seed 3 (out_weno_qwabl_s3)
(* w (+ 1.0 (* 0.3 (/ t (+ 0.000001 x)))))
```

Again all three are WENO-Z-family forms with no β_k/τ term, now with hand-picked round coefficients (0.15 and 0.3 on the power-1 form, 0.01 on the power-2 form). Measured transfer: out-of-distribution advection $+10.6 \pm 0.6\%$, Burgers $+47.5 \pm 3.7\%$, Euler $+28.8 \pm 1.5\%$ over WENO-Z at order 5.00, with all three seeds admissible (Euler overshoot at most 2.7×10^{-4}). These selected

constants satisfy the reported Euler threshold, whereas two calibrated open-weights winners exceed it.

Read together, the four arms separate what is reproducible from what is proposer-dependent: all nine frontier-proposer searches (six calibrated, three forward-only) select the β_k/τ correction, with four of the six calibrated seeds agreeing on the identical closed form of Equation 3, while all six open-weights searches settle on WENO-Z variants and never propose the term. At this budget the mechanism comes from the proposer; the constants that make a structure transfer, admissibly where the structure has headroom beyond dissipation, come from the evaluator.

Neural weight-network baselines, aggregate detail. The two trained weight networks of Section 5 (naive and order-preserving) are a sanity-check control, not a benchmark of the published constrained neural-WENO systems (Charles & Ray, 2025; Bezgin et al., 2022; Stevens & Colonius, 2020), whose guarantees are built in and which were not retrained here. They are not symbolic programs, so no program listing applies; their aggregate outcomes over three seeds each, on the identical objective and evaluation suite, are as follows. The naive network (1315 parameters) measures smooth order 3.06, out-of-distribution advection -36.9% (worse than WENO-Z despite training on advection), and raw Burgers $+58.4\%$ and Euler $+37.0\%$ attained with oscillation on every transfer problem ($0/3$ non-oscillatory; Burgers overshoot up to 6.1×10^{-2}), with across-seed spreads of ± 11 to 13%. The order-preserving variant measures order 3.33 (fourth order in one of three seeds), out-of-distribution advection -17.4%, raw Burgers $+53.7\%$ and Euler $+44.3\%$, one of three seeds non-oscillatory, and a Shu-Osher overshoot of 4.6×10^{-2}, larger than the discovered scheme's 1.49×10^{-2} where WENO-Z shows none. The advection-only non-oscillation penalty does not generalize for either network. The symbolic winners retain measured convergence order and meet the Burgers/Euler threshold, but that is not a general guarantee by construction.

D SOLVER VALIDATION AND BASELINE DETAILS

Every comparison in Section 5 runs on one of three solvers: a periodic linear-advection solver used for training, calibration, and the out-of-distribution test; an inviscid Burgers solver; and a one-dimensional Euler solver. Each solver was gated on a problem with a known solution before any scheme comparison, with each gate chosen to expose the implementation errors that solver could otherwise hide. The co-search itself runs the advection solver in single precision; every gate and comparison number in Section 5 and in this appendix is a float64 re-evaluation.

Weight functions and shared conventions. All schemes share the classical WENO5 components (Jiang & Shu, 1996; Shu, 1998): the three candidate stencil polynomials, the optimal linear weights $(d_0, d_1, d_2) = (0.1, 0.6, 0.3)$, and the Jiang-Shu smoothness indicators, which for the left-biased reconstruction at cell i are

$$\begin{aligned}
\beta_0 &= \tfrac{13}{12}\,(u_{i-2} - 2u_{i-1} + u_i)^2 + \tfrac{1}{4}\,(u_{i-2} - 4u_{i-1} + 3u_i)^2,\\
\beta_1 &= \tfrac{13}{12}\,(u_{i-1} - 2u_i + u_{i+1})^2 + \tfrac{1}{4}\,(u_{i-1} - u_{i+1})^2,\\
\beta_2 &= \tfrac{13}{12}\,(u_i - 2u_{i+1} + u_{i+2})^2 + \tfrac{1}{4}\,(3u_i - 4u_{i+1} + u_{i+2})^2,
\end{aligned} \tag{5}$$

with the global smoothness reference $\tau = |\beta_0 - \beta_2|$ (Borges et al., 2008). The three classical weight functions, in exactly the regularized forms our implementation evaluates, are

$$\text{WENO-JS:}\quad \alpha_k = \frac{d_k}{(\varepsilon_{\mathrm{JS}} + \beta_k)^2}, \qquad \varepsilon_{\mathrm{JS}} = 10^{-6}, \tag{6}$$

$$\text{WENO-Z:}\quad \alpha_k = d_k\left(1 + \left(\frac{\tau}{\varepsilon_{\mathrm{Z}} + \beta_k}\right)^2\right), \qquad \varepsilon_{\mathrm{Z}} = 10^{-12}, \tag{7}$$

$$\text{WENO-Z+:}\quad \alpha_k = d_k\left(1 + \left(\frac{\tau}{\varepsilon_{\mathrm{Z}} + \beta_k}\right)^2 + \eta\,\frac{\beta_k}{\tau + \varepsilon_{\mathrm{Z}}}\right), \qquad \eta = \Delta x^{2/3}, \tag{8}$$

each normalized as $\omega_k = \alpha_k / \sum_j \alpha_j$ (Jiang & Shu, 1996; Borges et al., 2008; Acker et al., 2016). The ε conventions matter at this accuracy level and are therefore fixed as above: 10^{-6} for WENO-JS following Jiang & Shu (1996), 10^{-12} for WENO-Z, and, deliberately, the same power 2 on the τ/β_k term and the same ε_{Z} for WENO-Z+ as for our WENO-Z, so that the appended $\eta\,\beta_k/(\tau + \varepsilon_{\mathrm{Z}})$ term

is the only difference between the two code paths and any Z+-versus-Z gap is attributable to that term alone. The discovered scheme (Equation 3) is evaluated from a fixed float64 transcription that regularizes both of its ratios with $\varepsilon = 10^{-6}$. The weight-function interface is uniform: a candidate receives the elementwise $(\beta_0, \beta_1, \beta_2)$ and returns the normalized weights, with τ and d_k formed inside, so classical baselines, neural networks, and evolved programs all pass through the identical solver code path.

Advection solver and its gate. The advection solver is a fifth-order finite-difference WENO reconstruction of the upwind flux on a periodic grid, integrated with three-stage SSP-RK3 (Gottlieb et al., 2001). The convergence gate advects $u_0 = \sin(2\pi x)$ and measures the L_1 error against the exact translate. Because SSP-RK3 is third order in time, a fixed CFL number would let the $O(\Delta t^3)$ temporal error swamp the fifth-order spatial error on fine grids; the gate therefore sets $\Delta t = 0.5\,\Delta x^{5/3}$ (CFL number $0.5\,\Delta x^{2/3}$), so the temporal error is $O(\Delta x^5)$ and time integration cannot mask a loss of spatial order. An initial sweep runs $N = 20$–320 over one period; the harness gate applied to every reported scheme runs $N = 40, 80, 160$ at $T = 0.2$ and reports the $\log_2$ error ratios of successive refinements as orders. In the harness gate the final time is snapped to a whole number of steps and the exact solution evaluated at the snapped time, so no terminal interpolation enters. Extending the gate one refinement further ($N = 40, 80, 160, 320$, same float64 protocol, $T = 0.2$, snapped final time), the discovered scheme measures orders 5.20, 5.11, and 5.04 under successive refinement (Section 5); WENO-Z measures 4.99, 5.00, 4.99 on the identical sweep. On the same smooth gate WENO-Z's absolute L_1 error is 2.1 to $2.6\times$ below the discovered scheme's ($2.56\times$ at $N = 40$, $2.22\times$ at $N = 80$, $2.06\times$ at $N = 160$), the purely-smooth concession stated in Sections 1 and 7. Two further checks complete the advection gate: a critical-point profile $\sin^2(2\pi x)$ at $N = 80$, where WENO-JS is known to lose accuracy at smooth extrema and WENO-Z must post the lower error (Borges et al., 2008); and a step advection test ($N = 200$, $T = 0.5$) recording overshoot, undershoot, and total variation against the exact value 2, with the unlimited linear weights included as the oscillatory control. The strict square-wave overshoot metric quoted in the results advects the indicator of $(0.3, 0.6)$ at $N = 200$ to $T = 0.4$ and reports $\max(u-1)_+ + \max(-u)_+$.

Burgers solver and its gate. The Burgers solver applies WENO5 to the Lax–Friedrichs flux splitting $f^{\pm} = \frac{1}{2}(f \pm \alpha u)$ with $f = u^2/2$ and $\alpha = \max_i |u_i|$ refreshed every step: f^+ is reconstructed with the left-biased stencils and f^- with a right-biased reconstruction that is the exact mirror of the left one about the $i + \frac{1}{2}$ interface (cell $i - k$ maps to cell $i + 1 + k$), using stencils $\{i+1, i+2, i+3\}$, $\{i, i+1, i+2\}$, $\{i-1, i, i+1\}$ with the same linear weights $(0.1, 0.6, 0.3)$, so one candidate weight function serves both biased reconstructions unchanged. Time stepping is SSP-RK3 with $\Delta t = \min(\mathrm{CFL}\,\Delta x/\alpha,\ T - t)$, terminating exactly at T. The gate integrates $u_0 = \sin(2\pi x)$ to $T = 0.1$, before the shock forms at $t = 1/(2\pi) \approx 0.159$, and compares against the exact characteristic solution of $u = \sin(2\pi(x - ut))$, solved pointwise by Newton iteration (80 iterations, valid for $t < 1/(2\pi)$); WENO-Z must converge at high order over $N = 40$–320 (CFL 0.3). This single gate exercises both biased reconstructions, the flux splitting, and the integrator at once: a sign or stencil error in the mirror reconstruction appears immediately as lost order. The transfer evaluation runs at $N = 100$, CFL 0.4: pre-shock error at $T = 0.1$ against the exact solution, and post-shock error at $T = 0.3$ against a WENO-Z reference computed on an $8\times$ finer grid and block-averaged in groups of eight cells onto the coarse grid.

Euler solver and its gate. The Euler solver ($\gamma = 1.4$) uses component-wise WENO5, the simpler standard choice at this problem scale (Shu, 1998), on a global Lax–Friedrichs splitting of the conservative flux with $\alpha = \max_i(|u_i| + c_i)$, the same mirrored left/right reconstructions as the Burgers solver, SSP-RK3 with the acoustic CFL condition, and transmissive boundaries implemented by three replicated ghost cells per side (a periodic variant serves the smooth gate). The gate advects a smooth density wave, $\rho_0 = 1 + 0.2\sin(2\pi x)$ with $u = 1$, $p = 1$, to $T = 0.2$ over $N = 40$–320 (CFL 0.3) and requires high-order L_1 convergence of the density against the exact translate. The Sod evaluation sets $(\rho, p) = (1, 1)$ for $x < 0.5$ and $(0.125, 0.1)$ for $x > 0.5$ at $N = 100$, $T = 0.2$, CFL 0.4, and reports per-field mean-squared errors for (ρ, u, p) against a fine-grid WENO-Z reference built by the same $8\times$ block-averaging protocol as for Burgers; the density overshoot metric is the excess above the left plateau 1 plus the deficit below the right plateau 0.125.

Table 8: **Absolute errors from the WENO-Z+ baseline run** (float64, identical evaluation harness for all rows). Advection is the mean-squared error of the Jiang-Shu composite profile (Gaussian, square, triangle, half-ellipse) after one full period at $N = 200$, against the exact translate; Burgers is the post-shock error at $T = 0.3$, $N = 100$, against the $8\times$ fine-grid block-averaged WENO-Z reference; Euler is the mean of the (ρ, u, p) mean-squared errors at the Sod tube, $t = 0.2$, $N = 100$, same reference protocol. WENO-Z+ uses its published $\eta = \Delta x^{2/3}$, recomputed per grid; the discovered row is the fixed seed-0 transcription of Equation 3. Lower is better; the improvement rows give the relative error reduction (higher is better).

Scheme	Advection (OOD)	Burgers (post-shock)	Euler (Sod)
WENO-JS (Jiang & Shu, 1996)	5.5802×10^{-3}	1.0295×10^{-3}	5.5679×10^{-4}
WENO-Z (Borges et al., 2008)	4.6348×10^{-3}	9.9256×10^{-4}	4.9625×10^{-4}
WENO-Z+ (Acker et al., 2016)	4.3200×10^{-3}	9.9102×10^{-4}	4.8825×10^{-4}
Discovered (seed 0)	3.2032×10^{-3}	5.2885×10^{-4}	3.0555×10^{-4}
WENO-Z+ vs. WENO-Z	$+6.8\%$	$+0.2\%$	$+1.6\%$
Discovered vs. WENO-Z+	$+25.9\%$	$+46.6\%$	$+37.4\%$
Discovered vs. WENO-Z	$+30.9\%$	$+46.7\%$	$+38.4\%$

The WENO-Z+ baseline. Because the discovered scheme shares the β_k/τ mechanism of WENO-Z+ (Acker et al., 2016), we run WENO-Z+ itself through the identical float64 evaluation harness behind Table 1. The implementation is Equation 8 with the published $\eta = \Delta x^{2/3}$ recomputed for every grid, since η is grid-dependent and WENO-Z+ is not self-similar: across this suite η ranges from 0.029 on the $N = 200$ advection grid through 0.046 on the $N = 100$ Burgers and Euler grids to 0.136 on the Shu-Osher grid ($\Delta x = 0.05$), which is exactly the dependence the discovered scheme's calibrated constant removes. WENO-Z+ passes the same gates it must: measured orders 5.15 and 5.09 on the $N = 40$–160 refinement, and a strict square-wave overshoot of 3.25×10^{-7}, matching WENO-Z's 3.32×10^{-7} in the same run, so its transfer showing below is not an implementation artifact. Table 8 reports the absolute errors underlying the relative rows of Table 1. WENO-Z+ improves on WENO-Z by $+6.8\%$, $+0.2\%$, and $+1.6\%$ on the three problems, while the discovered scheme beats WENO-Z+ head-to-head by $+25.9\%$ (out-of-distribution advection), $+46.6\%$ (Burgers), and $+37.4\%$ (Euler). As an internal consistency check, the discovered scheme's within-run improvement over WENO-Z ($+30.9\%$, $+46.7\%$, $+38.4\%$) reproduces its published seed-0 values, confirming the harness is identical; the main-text figures ($+34.0\%$, $+45.9\%$, $+40.6\%$) are means over six seeds. The same run includes the Shu-Osher stress test, reported in Appendix E.

Tuned expert-family baselines. The reviewer-standard control for a calibration claim: hand-specify the expert family and hand it the same calibrator. Each form below is written in the search's restricted surface (exponents as $e^{2c \log r}$ with $r = \tau/(\varepsilon + \beta_k)$; squared coefficients for positivity) and calibrated by the identical float64 protocol as every discovered scheme in Table 1: 50 Adam iterations at learning rate 0.03, two restarts from the same initialization distribution, the strict objective ($\delta = 3 \times 10^{-4}$, $\lambda = 100$). The grid-search control replaces gradients with a 10×10 grid over the β_k/τ coefficient $A \in [10^{-2}, 10]$ (log-spaced) and the WENO-Z exponent $p \in [1, 3]$, Z coefficient fixed at its published 1: exactly 100 objective evaluations, matching the 50×2 budget of the gradient path, winner selected on the training objective alone.

Table 9: Tuned expert-family baselines (improvement over WENO-Z, identical evaluation suite). The full family matches the discovered scheme; the tuned WENO-Z family does not reach its out-of-distribution gain from tuning alone.

Form (free knobs)	Adv. OOD	Burgers	Euler	Order	Constants found
WENO-Z family, free coeff. + exponent	$+12.3\%$	$+58.4\%$	$+29.8\%$	5.0	b^2=0.012, p=1.62
WENO-Z+, free η only	$+32.9\%$	$+12.7\%$	$+22.6\%$	5.1	a^2=5.30
Full family (damped Z, free p, free β_k/τ)	$+39.6\%$	$+45.6\%$	$+38.6\%$	5.1	a^2=1.64, b^2=0.23, p=1.39
Grid search over (A, p), no gradients	$+43.3\%$	$+43.7\%$	$+41.6\%$	5.1	A=10, p=1.44

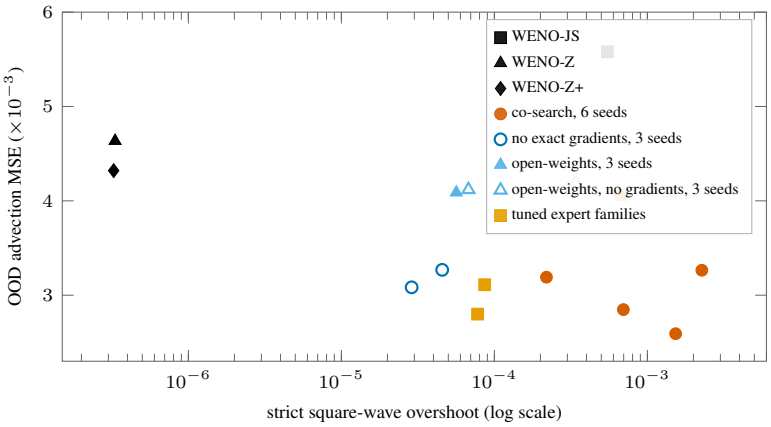


Figure 4: **Accuracy against oscillation for every scheme and seed: a design frontier, not a single winner.** Out-of-distribution advection MSE (lower is better) against the strict square-wave overshoot (the harshest oscillation probe; log scale), all searched and tuned schemes recalibrated in float64 by the identical published protocol. WENO-Z and WENO-Z+ hold the extreme low-overshoot corner at the highest error; every search arm and tuned family buys accuracy by admitting more overshoot, and the full co-search populates the low-error frontier. Structurally identical seeds calibrate identically, so coincident points overlap (the six full-arm and three open-weights seeds draw as four and two points). The trained neural weight networks sit far outside the frame (overshoots 4.6×10^{-2} to 6.1×10^{-2} with OOD error worse than WENO-Z). Data: `results/pareto_points.csv` in the supplement.

Three details temper the reading. The grid winner sits at the A=10 search boundary, so the two-knob slice may extend further; the budget-matching, not the boundary, is the controlled quantity. The full family lands at a different optimum within the family (b^2=0.23, p=1.39) than the discovered scheme ($b^2 \approx 0.038$, p=2): the family carries a ridge of comparable solutions and systematic calibration reaches it from either parameterization. And all four arms are admissible (Burgers and Euler overshoots below 10^{-3}; the largest, tuned WENO-Z's 3.4×10^{-4} on Euler, echoes the open-weights arms' pattern of Z-family tuning spending the non-oscillation margin, here without crossing it). Grid search is exponential in the number of knobs, so exact gradients remain the calibration that scales to many-constant programs; at two knobs, exhaustion suffices, and the conclusion of Section 5 is unchanged: the value the co-search adds is arriving at the family, and at calibrated constants, without being told.

Two-dimensional advection check. A dimension-by-dimension extension of the advection solver (velocity $(1, 1)$, periodic, SSP-RK3 with the two-dimensional CFL sum) evaluates the frozen schemes without recalibration. Smooth convergence (product sine, $N = 40$ to 160): WENO-JS $5.00/5.00$, WENO-Z $5.00/5.00$, WENO-Z+ $5.04/5.01$, discovered $5.06/5.02$; fifth order holds in two dimensions for all schemes. On a discontinuous transfer (a Gaussian bump, a square patch, and a cone advected one period at $N = 200^2$) the discovered scheme's accuracy edge persists, MSE 1.334×10^{-3} against WENO-Z's 2.057×10^{-3} (+35.2%) and WENO-Z+'s 1.903×10^{-3}, but its overshoot grows to 4.19×10^{-3} where WENO-Z admits 7.8×10^{-7} and even WENO-JS only 1.19×10^{-3}: the one-dimensional margin thins under dimension-by-dimension reconstruction, the interaction the limitations section names, now quantified. Restoring the margin would require the non-oscillation constraint enforced in a two-dimensional objective.

The accuracy-oscillation frontier across all schemes. Figure 4 plots every scheme and seed of the study on the accuracy-versus-oscillation plane: out-of-distribution advection error against the strict square-wave overshoot, the harshest oscillation probe (the training constraint is enforced on the training mix; this square test is evaluated post hoc). No scheme dominates: the classical schemes hold the extreme low-overshoot corner at the highest error, the no-gradient arms sit at conservative low-overshoot points with part of the accuracy, and calibration moves schemes along the frontier toward lower error. This is the tradeoff of Section 5 made visible: the co-search selects a point on a design frontier under the constraint it is given, rather than a scheme that wins everywhere.

An expert-family accuracy–robustness tradeoff. WENO-Z+M (Luo & Wu, 2021) modifies the reversed-ratio correction with an adaptive coefficient. Under the recorded evaluation it improves on WENO-Z+ by 14.6%, 38.0%, and 29.8% on advection, Burgers, and Euler, respectively. The discovered scheme has lower smooth and out-of-distribution error in these comparisons, but WENO-Z+M is more robust on the stronger shock.

The additional-term exponent e illustrates why fitting accuracy alone can be misleading (Figure 3 in the main text). Its published value is 0.5. Jointly fitting the coefficient and exponent on the smooth training objective moves the exponent to about 1.43, improving smooth transfer while producing a Shu–Osher range violation of about 0.64, versus 6×10^{-3} at the published setting. A separate sweep fixes the base coefficient at 579 and gives a violation of about 1.83 at $e = 1.43$; this is the curve in the figure. The range violation rises sharply between $e = 1.2$ and $e = 1.3$, and evaluations diverge at $e \geq 1.5$. The discovered expressions retain exponent 1.0, and two include additional damping terms. These examples show how fixing part of a formula can restrict a damaging optimization direction. They do not establish that structure search guarantees robustness: all six primary winners have nonzero Shu–Osher undershoot in the independent-reference evaluation (Appendix F), and the expert-family sweep optimizes a different, smooth objective.

E SHU-OSHER STRESS TEST IN DETAIL

Setup. The Shu-Osher shock-entropy problem (Shu & Osher, 1989) advances the one-dimensional Euler equations with $\gamma = 1.4$ on $x \in [-5, 5]$ to $t = 1.8$, starting from a Mach-3 shock at $x = -4$ running into a smooth high-frequency density field:

$$(\rho,\, u,\, p) \;=\; \begin{cases} (3.857143,\; 2.629369,\; 10.33333) & x < -4, \\ (1 + 0.2\sin(5x),\; 0,\; 1) & x \geq -4. \end{cases}$$

The passing shock compresses and amplifies the entropy waves, so the post-shock region carries fine oscillatory structure that a dissipative scheme smears, while the shock itself punishes any scheme that under-damps; resolving both at once is the property WENO schemes exist to provide. All schemes run through the same validated solver as the Sod transfer test of Section 5: component-wise WENO5 reconstruction with global Lax-Friedrichs flux splitting, third-order SSP Runge-Kutta (Gottlieb et al., 2001) at CFL 0.4 with the time step recomputed each step from the maximum wave speed, transmissive boundaries via three replicated ghost cells per side, and float64 arithmetic throughout. The evaluation grid has $N = 200$ cells ($\Delta x = 0.05$). The reference is the same solver with WENO-Z weights on an $8\times$ finer grid ($N = 1600$), block-averaged onto the coarse grid by eight-cell means; this is the same fine-grid WENO-Z construction used for the Burgers and Euler references in Section 5, standard and if anything slightly favorable to the WENO-Z baseline. We report the coarse density's mean-squared error against this reference both over the full domain and in the post-shock high-frequency window $x \in [0.5, 2.5]$, and an envelope violation defined as the maximum excursion of the coarse density above the reference maximum plus its maximum excursion below the reference minimum; for every scheme here the violation, where present, is an undershoot below the reference minimum, which lies in the undisturbed incoming field.

Results. Table 10 reports the discovered scheme of Equation 3 against WENO-JS and the two Z-family baselines, all evaluated in one deterministic float64 run. In the high-frequency window the discovered scheme attains the lowest error of the four, 2.9% below WENO-Z (Borges et al., 2008), and pays for it with the only non-negligible undershoot, 1.49×10^{-2} below the incoming density field (about 1.5% of that field's mean density); densities remain positive everywhere. WENO-Z+ (Acker et al., 2016), with its published grid-dependent $\eta = \Delta x^{2/3}$ recomputed for this grid, moves in the same direction but far more timidly: its high-frequency error sits 2.6% below WENO-Z with a negligible undershoot of 8.8×10^{-5}, and it recovers correspondingly less of the post-shock structure. WENO-JS (Jiang & Shu, 1996), the most dissipative of the four, sits 5.4% above WENO-Z in the window; neither it nor WENO-Z shows any undershoot. On whole-domain mean-squared error the ranking shifts: the discovered scheme (3.2101×10^{-2}) sits slightly above both WENO-Z (3.1766×10^{-2}) and WENO-Z+ (3.0936×10^{-2}), while WENO-JS trails all three (3.3662×10^{-2}). The full-domain metric is dominated by the shock and contact regions rather than the entropy waves, so this is the wash at the extreme made explicit: the discovered scheme trades a small amount

Table 10: **Shu-Osher stress test** ($N = 200$, $t = 1.8$, float64, single deterministic run). Density mean-squared error against the block-averaged fine WENO-Z reference, over the full domain and in the post-shock high-frequency window $x \in [0.5, 2.5]$, and the undershoot below the reference envelope. The discovered scheme resolves the most post-shock structure but admits the only non-negligible undershoot and sits slightly above the Z-family baselines on whole-domain error; WENO-Z+ moves in the same direction far more timidly.

Scheme	Full-domain MSE	High-freq. MSE	High-freq. vs Z	Undershoot
WENO-JS (Jiang & Shu, 1996)	3.3662×10^{-2}	1.6387×10^{-1}	$+5.4\%$	0
WENO-Z (Borges et al., 2008)	3.1766×10^{-2}	1.5542×10^{-1}	(reference)	0
WENO-Z+ (Acker et al., 2016)	3.0936×10^{-2}	1.5141×10^{-1}	-2.6%	8.8×10^{-5}
Discovered (Equation 3)	3.2101×10^{-2}	1.5094×10^{-1}	-2.9%	1.49×10^{-2}

of whole-domain accuracy at this strong shock for the best resolution of the amplified post-shock structure.

Total variation, under two conventions. Total variation of the coarse density summarizes how much of the amplified post-shock structure a scheme retains, but its numeric value depends on the summation convention, so we report the two conventions used by our tooling separately and attribute each. The standalone shock diagnostic sums $|\rho_i - \rho_{i-1}|$ non-cyclically over the domain and gives WENO-Z 8.1 and the discovered scheme 13.1 against a reference of 19.2: the discovered scheme retains roughly two thirds of the reference's variation where WENO-Z retains roughly 42%. The baseline evaluation driver behind Table 10 instead sums cyclically over the whole domain, which adds the wrap-around jump between the two domain ends to every value; under that convention the same solutions give WENO-Z 11.1, the discovered scheme 16.0, and a reference of 22.1. The reference in both cases is the same block-averaged fine-grid solution, summed once under each convention; the wrap-around jump of about 2.9 accounts for the difference in every pair. Values must not be compared across the two conventions. Under both, the discovered scheme sits well above WENO-Z and below the reference: it recovers a substantial part of the structure WENO-Z dissipates without manufacturing variation beyond the reference.

Reading. These measurements quantify the reading already given in Section 5: at this strong shock the discovered scheme is not a clean win but a less-dissipative point on the accuracy-robustness frontier, strongly favorable on the moderate-shock suite and roughly a wash at the extreme.

F WENO EVALUATION CONVENTIONS AND NUMERICAL CHECKS

The original tables use the evaluation drivers and references recorded with the search campaign. This appendix makes the effect of those conventions explicit using an existing follow-up over the same selected programs. No new search is needed. The supplement contains all 24 numerical records, 33 regularization-specific evaluations, reference arrays, and their analysis code under `experiments/review_gates/`.

Why the reference matters. A pointwise finite-difference solution should ideally be compared with the solution at those same points. The original Burgers and Sod evaluations instead averaged blocks of a finer-grid WENO-Z solution. Near a discontinuity, a block average and a point value need not agree. Independently constructed entropy/Riemann references and matched regularization therefore provide a useful check. The small positive constant ε prevents division by zero in weight formulas; matching it avoids attributing a regularization change to the selected program.

Table 11: Mean error reduction versus WENO-Z for the six primary WENO winners. The first row preserves Table 1's reporting convention. The second uses independent Burgers/Sod point references and matches $\varepsilon = 10^{-6}$ in candidates and baseline.

Evaluation convention	Advection	Burgers	Sod
Original reporting	34.0%	45.9%	40.6%
Matched regularization, independent references	34.1%	40.9%	32.2%

The improvement persists, but its magnitude is sensitive to reference construction. The matched primary advection solutions also have range violations of about 0.009–0.024, despite low Burgers/Sod violations. Consequently the original table's "Non-osc." column, defined only by Burgers and Euler checks, must not be read as a guarantee on every transfer problem. On the matched Shu–Osher comparison, mean whole-domain error is about 10% worse than WENO-Z and all six solutions retain an undershoot. These observations qualify broad robustness claims.

When calibration occurs. Four calibrated-search winners and all three uncalibrated Fable winners can be expressed as the same two-coefficient family after freeing both correction coefficients. Under an identical post-selection fit, that family receives identical fitted coefficients and performance. The original gradient/no-gradient comparison therefore establishes a difference between the reported workflows; it does not isolate a causal advantage of fitting before structural selection. The two other primary winners contain additional damping structure. Their expressions, fitted coefficients, failures, and complete numerical arrays remain available in the supplement.

G A SECOND TASK: STATE-SPACE MODEL STRUCTURE

This task tests language features that the WENO weight expressions do not exercise. A Kalman filter maintains a hidden state and updates its prediction after each observation. The program loops over T observations, uses branches to switch the noise model, and performs matrix operations in each update (`matvec`, `matmul`, `transpose`, `inv`, `logdet`, `dot`). The design fixes a validated filter template and lets the proposer write only the model: the state dimension $n \in \{1, 2, 3\}$ and six fragments constructing the initial state and covariance (X_0, P_0, screened but not structure-defining) and the state-space matrices (F, H, Q, R) in a restricted matrix-construction surface (`mat`, `vec`, `eye`, `scale`, `transpose`, scalar arithmetic, comparisons inside `if`, free scalar parameters `a` to `d`, and the previous observation read as `(ref yp 0)`). The filter recursion, which accumulates the per-step innovation likelihood $\log\det S_k + e_k^\top S_k^{-1} e_k$, is spliced around the fragments by the harness and is never editable. A static screen enforces the surface, shape consistency across fragments, a size cap, and one rule inherited from the gradient scoping of Section 3: every `if` predicate must be parameter-free, since a threshold written as a calibrated parameter would silently never move; a corollary is that control flow is uniform across restart lanes, so one structural walk of the template serves every restart of a candidate. Variance-like entries follow a squared-parameter convention, `(* a a)`, so Q and R stay positive semidefinite without clamps. The scope caveat, pre-committed before any run: this task demonstrates that the same co-search machinery operates on a loop, branch, and matrix program class; it is a breadth demonstration with baselines, not a state-of-the-art claim in time-series model selection.

Two validation gates preceded any campaign. First, every matrix program in the oracle suite of Appendix B was square, while this template is rectangular (H is $1 \times n$, with a one-by-one innovation covariance, inverse, and log-determinant), so the conformance suite was extended with general-observation-map filter programs at $n = 1, 2, 3$ and re-run before any search: 36 of 36 forward and 88 of 88 gradient comparisons against the eager oracle, 10 of 10 batched-versus-oracle checks, and 66 equivalence tests pass. Second, an end-to-end harness gate ran one hand-written candidate per target and required a finite forward value, nonzero gradients on every free parameter, and a falling likelihood under calibration; all four cells pass. At the campaign's inner budget, 8 Adam restarts of 150 steps each (learning rate 0.05) batched as lanes of one structural walk, a candidate calibrates in 4.50 s. The inner objective is the train-window filter negative log-likelihood summed over replicates; the outer score is held-out NLL per step, obtained as the difference of two template evaluations of the same candidate at loop bounds $T = 200$ and $T = 150$ (exact, since both traverse the same prefix), and restarts are selected on the train window only, so multistart never reads the scoring split. The campaign is four synthetic targets by three seeds at 200 outer iterations, the outer machinery of Appendix A otherwise unchanged; all twelve runs completed in 3 h 28 min. Two additional controls probe the proposer and calibration separately: a random-proposal arm that draws model blocks from a grammar frozen before any control run (state dimension uniform on $\{1, 2, 3\}$; transition families spanning identity, diagonal, companion, dense, and the trend block; isotropic, diagonal, or rank-one process noise; a plain or observation-switched noise term whose threshold literal is drawn log-uniform from $[0.5, 1.5]$, deliberately wider than the registry accepts) and calibrates each through the same

evaluator path at the same 200-proposal budget, isolating the proposer; and a no-gradient ablation with parameters frozen at 1.0, isolating the calibrator.

Targets and pre-registration. The four targets use $T = 200$ steps (train 150, holdout 50), observation dimension one, and three replicate datasets each under fixed generation seeds. S1 is the local level model, $x_{t+1} = x_t + w_t$, $y_t = x_t + v_t$ with $\sigma_w = 0.3$, $\sigma_v = 0.5$: the classical menu's simplest member and the sanity anchor. S2 is the local linear trend, level plus slope with $F = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$, $H = [1\ 0]$, level noise 0.10^2, slope noise 0.05^2, observation noise 0.5^2: it requires $n = 2$ and the off-diagonal unit. S3 is an AR(2) in companion form, $\phi_1 = 1.5$, $\phi_2 = -0.8$ (a stationary pseudo-cycle of period about 11 steps), rank-one process noise 0.5^2 on the driven state, observation noise 0.3^2. S4 is deliberately off the menu: a threshold-heteroskedastic AR(1) level, $x_{t+1} = 0.9x_t + w_t$ with $\sigma_w = 0.4$, observed under noise whose standard deviation is 0.15 when the previous observation satisfies $|y_{t-1}| \leq 1.0$ and 0.90 otherwise. Because the regime indicator is a deterministic function of observed data, the switched filter is still the exact likelihood, so S4 is a fair likelihood target; but no member of the frozen classical menu (the structural models local level, local linear trend, level plus AR(1) cycle, and level plus damped trend, and the SARIMAX models AR(1), AR(2), ARMA(1,1), and ARIMA(1,1,0)) admits observation-driven variance switching, so menu selection cannot recover it even in principle, while the generating switch is writable in the fragment surface as an `if` on `(ref yp 0)`. Generation enforces that both regimes are visited (at least 20% occupancy; the realized quiet-regime occupancy is 0.69 to 0.79).

Recovery is judged against a registry frozen before any campaign output existed. Tier-1, the headline, requires a canonical-form match (fragments are parsed, `let*` bindings inlined, literal subtrees folded, parameters renamed by first appearance; accepted orientation variants are enumerated per target; X_0 and P_0 are excluded because the filter forgets them) plus parameter recovery: 20% relative bands by default, structural slots pinned at 1.0 accepting ± 0.05 (the tight unit-root band is what separates S1's $F = 1$ from S4's $\phi = 0.9$), and for S4 a threshold literal in $[0.75, 1.25]$ with the two variances correctly ordered across the branches. Tier-2, behavioral, requires the correct state dimension, held-out NLL within 3 nats total of the generating model's, and one-step forecast RMSE within 5% of the truth's, on every replicate; it exists because state-space models are identified only up to similarity transforms of the state. A structure-only diagnostic (all structural checks, value bands skipped) is reported alongside. After the freeze, but recorded before any campaign output was read, calibrating the exact truth forms on the generated replicates showed that two registered bands are unattainable or marginal on these data: the S2 trend form's level-noise MLE is 0.0338 against a registered band around 0.01 (3.4 nats better than the generating values on the train window; the $T = 150$ variance decomposition is weakly identified), and the S4 process-noise MLE, about 0.196, sits at the band edge 0.192. The decision, recorded at the time: the registry stays frozen, Tier-1 is reported exactly as registered, and the structure-only and Tier-2 columns carry the distinction, with this attainability analysis as the documented explanation; no bands were amended.

Results under the frozen registry. Table 12 gives every verdict, computed by one analyzer against the registry exactly as frozen. Across the main arm, Tier-2 passes eight of thirteen runs (twelve campaign seeds plus the preflight); every dimension-inflated candidate fails it through the state dimension.

On S3 the search recovers the registered structure in full. Seed 3's winner, found at iteration 139 of 200, is quoted verbatim from its output directory (`out_kalman_S3_ar2_companion_s3`):

```
STATE_DIM = 2
X0 = (vec 0.0 0.0)
P0 = (scale (* d d) (eye 2))
F = (mat (vec a b)
         (vec 1.0 0.0))
H = (mat (vec 1.0 0.0))
Q = (mat (vec (* c c) 0.0)
         (vec 0.0 0.0))
R = (mat (vec (* d d)))
```

Calibration fits $a = 1.463$ (ϕ_1, truth 1.5), $b = -0.763$ (ϕ_2, truth -0.8), $c^2 = 0.259$ (process noise, truth 0.25), and $d^2 = 0.088$ (observation noise, truth 0.09): every slot inside its registered band, in

Table 12: Structure recovery under the frozen registry, matches out of three seeds per target, across the full proposer (frontier Fable / open-weights Qwen) by calibration (exact gradients / no gradients) two-by-two, plus random-proposal and classical AIC/BIC controls. Each arm reports Tier-1 (full match) / Form (structure-only diagnostic, parameter-value bands skipped) / Tier-2 (behavioral). A 25-iteration preflight on S1, before the campaign, returned the exact registered local-level block, a full Tier-1 match, not counted in the S1 Fable cell. The menu column reports which model BIC selected per replicate; the "damped trend" member is realized as a local level plus a damped stochastic cycle; S4 is outside the menu by construction, so its picks are the best homoskedastic compromise.

Target	+ exact gradients		no gradients			
	Fable	Qwen (open)	Fable	Qwen (open)	Random	Menu (BIC)
	Tier-1 / Form / Tier-2, of 3					
S1 local level	0 / 0 / 1	0 / 0 / 2	0 / 0 / 1	**3 / 3 / 3**	0 / 0 / 0	local level 3/3
S2 linear trend	0 / 1 / 3	0 / 0 / 3	0 / 3 / 3	0 / 2 / 2	0 / 0 / 3	trend 2/3, LL 1/3
S3 AR(2) companion	**1** / 1 / 3	0 / 0 / 0	**1** / 1 / 1	0 / 0 / 0	0 / 0 / 0	AR(2) 2/3, d. trend 1/3
S4 threshold (off-menu)	0 / 0 / 0	0 / 0 / 2	0 / 0 / 0	0 / 0 / 0	0 / 3 / 3	LL 2/3, AR(2) 1/3

the registered companion orientation. The reuse of $\mathtt{d}$ in P_0 is immaterial, since P_0 is not structure-defining and is excluded from the match. The other two S3 seeds recover the same companion dynamics with coefficients near the truth (ϕ pairs $1.45/-0.75$ and $1.453/-0.752$ against $1.5/-0.8$) but fail Tier-1 on spurious observation-noise switches, quoted below.

On S2 all three seeds find the trend structure: each builds $F = \left[\begin{smallmatrix} 1 & 1 \\ 0 & \phi \end{smallmatrix}\right]$ with ϕ inside the unit-root band (two as calibrated parameters, one as the literal 1.0) and $H = [1\ 0]$. None passes Tier-1, and the failure is exactly the one the pre-freeze attainability note documented: the registered level-noise band cannot be met on these replicates (the truth form's own MLE lands at 0.0338 against the band around 0.01), and the discovered blocks make the same trade in a more extreme form, pinning the level noise to a literal zero and letting the slope noise carry the variability. Seed 1 is otherwise structurally exact and fails only that value band; seeds 2 and 3 additionally carry an observation-noise switch the truth lacks, which is itself a structural miss, so the Form column credits one seed, not three.

S1 is the mixed outcome. The 25-iteration preflight returned the exact registered block, with both variances in band (0.081 against truth 0.09; 0.221 against truth 0.25). The three 200-iteration campaign seeds kept searching past that point: seed 1 keeps $n = 1$ and correct level dynamics (its transition slot calibrates to 0.986, inside the unit-root band) but replaces the constant observation variance with a two-branch switch whose loud branch scales with $|y_{t-1}|$, a Tier-2 pass and a Tier-1 miss; seeds 2 and 3 inflate the state to dimensions 2 and 3 and miss at every tier. On the menu's own simplest member, BIC recovers the local level on all three replicates: longer adaptive search on this target buys held-out likelihood through structure the truth does not have.

S4 is the registered negative, reported as registered. All three seeds return two-state models: each carries a threshold switch on the previous observation at the generating threshold, but its quiet branch collapses to an observation variance near zero (fitted $b \approx 10^{-4}$, squared), so a second latent state, not the registered one-state two-variance switch, carries the heteroskedasticity. All three are misses through the state dimension, at raw held-out NLL -0.499 to -0.493 per step, far better than the truth form fits. Pre-registered structure recovery does not follow from excellent fit when an equally expressive alternative parameterization exists. The menu's best homoskedastic compromise on the same cells scores -0.12 to 0.29 per step. Across all cells, BIC's picks appear in the table; AIC, which penalizes structure less, instead picks the level plus damped trend member on five of the twelve replicates where BIC does not.

The registry promised every Tier-1/Tier-2 disagreement verbatim; there are six, and their model blocks (the structure-defining fragments) follow, quoted from the analyzer's listing:

```
;; S1 seed 1: correct level block, switched R (transition slot d
    fitted 0.986)
F = (mat (vec d))
H = (mat (vec 1.0))
Q = (mat (vec (* a a)))
R = (mat (vec (if (<= (abs (ref yp 0)) 1.0) (* b b) (* c (* c (abs (
    ref yp 0)))))))
```

```
;; S2 seed 1: exact trend block, level noise pinned to zero, plain R
F = (mat (vec 1.0 1.0) (vec 0.0 d))
H = (mat (vec 1.0 0.0))
Q = (mat (vec 0.0 0.0) (vec 0.0 (* a a)))
R = (mat (vec (* b b)))

;; S2 seed 2: trend block, zero level noise, switched R
F = (mat (vec 1.0 1.0) (vec 0.0 a))
H = (mat (vec 1.0 0.0))
Q = (mat (vec 0.0 0.0) (vec 0.0 (* c c)))
R = (mat (vec (if (<= (abs (ref yp 0)) 1.0) (* b b) (* d d))))

;; S2 seed 3: trend block with literal unit roots, zero level noise,
    switched R
F = (mat (vec 1.0 1.0) (vec 0.0 1.0))
H = (mat (vec 1.0 0.0))
Q = (mat (vec 0.0 0.0) (vec 0.0 (* a a)))
R = (mat (vec (if (<= (abs (ref yp 0)) 1.0) (* b b) (* d d))))

;; S3 seed 1: companion dynamics as literals, two-threshold nested
    switch on R
F = (mat (vec 1.4503 -0.7497) (vec 1.0 0.0))
H = (mat (vec 1.0 0.0))
Q = (mat (vec (* c c) 0.0) (vec 0.0 0.0))
R = (mat (vec (if (<= (abs (ref yp 0)) 1.0) (* b b)
          (if (<= (abs (ref yp 0)) 2.0) (* d d) (* a a)))))

;; S3 seed 2: companion dynamics (c fitted 1.453), nested switch on R
F = (mat (vec c (- 0.0 0.7517)) (vec 1.0 0.0))
H = (mat (vec 1.0 0.0))
Q = (mat (vec (* a a) 0.0) (vec 0.0 0.0))
R = (mat (vec (if (<= (abs (ref yp 0)) 1.0) (* b b)
          (if (<= (abs (ref yp 0)) 2.0) (* d d) (* 2.25 (* d d))))))
```

What the controls establish. A good likelihood score does not necessarily identify the generating mechanism. On S1, all three random-proposal winners are structurally incorrect two-state models, yet their held-out NLL of 0.086–0.089 per step is lower than the main campaign's 0.102–0.107. The S1 seed-1 campaign winner also beats the generating model's likelihood on two replicates by 2.7 and 1.6 nats. These are finite-sample scores on data repeatedly used for selection. They are not evidence that the recovered mechanisms are more accurate descriptions of the generator.

Five of the six Tier-1/Tier-2 disagreements above introduce an observation-noise switch absent from the truth. Their forecast errors remain close to the generating model's, so prediction alone does not resolve the structural ambiguity. The frozen registry makes that distinction visible. On S4, random proposals recover the generating form in all three seeds but never meet the full parameter-recovery criterion. The random grammar explicitly contains the switch primitive; its form-recovery result depends on that prior choice of building blocks.

The proposer/calibration comparison is also mixed. Fable is the only tested proposer to achieve full AR(2) recovery, with one success both with and without gradients. For the simpler S1 target, the open-weights no-gradient arm instead recovers the local-level model in all three seeds. For S2, removing calibration increases form recovery from one to three Fable seeds, while full parameter recovery remains absent. These outcomes demonstrate reusable execution and fitting of structured programs, but do not establish that longer search or gradient calibration consistently improves mechanism identification.

Scope and harness notes. Three harness facts are documented rather than repaired mid-campaign. First, the deployed screen admits `let*` in its allowlist but rejects any bound name outside the advertised variables as a foreign token, so let bindings are effectively unusable in proposals even though the prompt mentions them; a proposal that used one wasted its iteration and received the rejection reason as an artifact, and the screen was deliberately left unchanged once the campaign was running. Second, the classical menu is scored on the same split and mapped into the template's

held-out NLL convention so the columns compare directly, and one frozen-menu entry has no exact statsmodels equivalent: "level plus damped trend" is implemented as local level plus a damped stochastic cycle, with the mapping recorded in the baseline's output. Third, five points the frozen registry left open (parameter sharing across slots; the band for zero-pinned observation-map entries; slot order under the swapped trend orientation; one-sided Tier-2 bands, with candidates beating the truth beyond the band flagged for audit rather than failed; and Tier-2 required on all three replicates) are resolved one way in the analyzer and recorded with its output. The pre-registered real-data island, the ENSO Nino3.4 monthly index scored on held-out NLL and one-step forecast skill with no structure oracle, was not run in this revision. All available program, prompt, and score records ship with the supplement archive, alongside the frozen registry, generated targets, and analyzer that produced Table 12. Checkpoint eviction and responses discarded before persistence limit the recovered history; the artifact inventory identifies the records that are available.

H Where Exact Gradients Matter: Optimizer Scaling

This diagnostic isolates the value of gradient information from the choice of runtime. Both the WENO family and dense state-space filter are implemented directly in PyTorch, whose autograd supplies the derivatives. We vary the number k of free constants and compare several optimizers at a fixed objective-evaluation budget: Adam takes 50×2 steps, and each derivative-free baseline (Nelder-Mead, Powell, differential evolution, and random search) is given 200 forward objective evaluations, twice the forward solves the gradient path performs. Treating a reverse pass as one additional forward solve is a budget convention; the experiment does not establish equal wall time or equal computational cost across optimizers.

On the WENO weight family (Table 13) the crossover is muted. The advection-fit objective is low in effective dimension, one WENO-Z-like correction largely suffices, so exact gradients stay stable across k but do not dominate: the global derivative-free methods keep pace at every k, and Nelder–Mead and Powell have much worse final objectives at k=20. This diagnostic therefore does not establish a consistent gradient advantage on the chosen WENO family. On a dense state-space likelihood (Table 4), where the free entries of F, Q, and R are many and coupled, the difference grows with dimension: Adam, Nelder–Mead, and Powell are close through k=4, but by k=10 exact-gradient calibration reaches a training negative log-likelihood of -84.1 while the derivative-free methods attain higher final objectives ($+39.7$, $+46.1$, -25.0, $+14.8$), a gap of order 100. This is evidence for gradient-based optimization on these fixed families. The changing-program state-space search uses NDVM, but this diagnostic does not compare its derivative implementation or execution time with PyTorch.

Table 13: WENO weight family: best training objective ($\times 10^{-3}$, lower is better) per optimizer as the free-constant count k grows. Exact gradients are stable; Nelder–Mead and Powell have larger objectives at k=20; global derivative-free keeps pace.

k	Adam (exact grad.)	Nelder-Mead	Powell	Diff. evolution	Random
2	1.34	1.23	1.56	1.36	1.37
5	1.51	1.24	1.21	1.36	1.49
10	1.81	2.03	1.88	1.76	1.70
20	1.98	638	147	2.00	1.83

The dense state-space diagnostic tests k=2, 4, 6, 10; Table 4 shows all four settings, and the supplement contains every result. It measures optimizer quality on a fixed 80-step sequence, with no train/holdout split. Its objective sums $\log\det S_t + e_t^\top S_t^{-1} e_t$, where e_t and S_t are the filter's prediction residual and innovation covariance. This is twice Gaussian negative log-likelihood with the additive constant omitted; the convention is shared by all methods in the table.

I A classical symbolic-regression baseline

This control asks whether fitting examples of a weight function is sufficient to recover a useful solver component. PySR (Cranmer, 2023) fits the enhancement factor $h = \alpha_k / d_k$ from inputs (β_k, τ) using an arithmetic grammar. Training pairs are sampled from real solver states. We then evaluate the

selected expression through the fluid solver, both with its original constants and after differentiable recalibration. The important distinction is the training objective: pointwise regression matches supplied weights, whereas our co-search scores the solution produced by a full simulation.

We supply three targets: WENO-Z, published WENO-Z+, and a WENO-Z+ variant with an amplified reversed-ratio term. The last target makes that mechanism easier to detect. None of the selected expressions, or the recorded expressions on PySR's complexity–error frontier, contains the β_k/τ mechanism under the diagnostic used in the experiment. The selected rational expressions contain large constants. Two produce finite transfer evaluations but have advection errors about 5–6% worse than WENO-Z; the third produces a nonfinite result. Recalibration leaves the finite advection results worse than WENO-Z and makes a second expression nonfinite. The supplement preserves the targets, every recorded frontier expression, the mechanism probe, and both raw and recalibrated evaluations.

These results show a mismatch between good pointwise regression and useful solver behavior in this setup. They do not show that symbolic regression cannot recover the mechanism, or that a symbolic search using the same simulation-based objective would fail. The language-model and PySR runs use different objectives and proposal budgets, so this is a diagnostic of the chosen pointwise formulation rather than a matched comparison of proposer efficiency.

J GUIDE TO THE ANONYMOUS ARTIFACT

The supplementary ZIP accompanies this manuscript and includes its LaTeX source. Start with `README.md`; `REPRODUCE.md` maps the paper's claims to saved evidence and commands. The package separates three tasks that require different resources: checking saved results, running numerical checks on fixed programs, and regenerating a search with a language-model service. Reading the evidence and running the first two tasks do not require new model requests.

Restore and check the records. Run `python3 restore_artifact.py` after extraction. Repeated checkpoint files are stored once, and large text records are losslessly compressed; the restoration script verifies hashes before recreating their paths. Then run `python3 tools/verify_artifact.py` and `python3 tools/verify_evidence.py`. These checks inspect file integrity and reconstruct reported summaries from saved records. They do not execute proposed Python wrappers or rerun the searches. The manifest identifies source variants, anonymization, and the available record coverage. All recovered rejected proposals remain included; missing pre-checkpoint history is not reconstructed or silently counted as successful.

Locate an experiment. `experiments/exp_weno/` contains the WENO solvers, proposed programs, settings, replay tools, neural controls, and saved results. Its `optscaling/` and `srbaseline/` subdirectories contain the optimizer and PySR diagnostics. `experiments/exp_kalman/` contains the state-space targets, frozen recovery registry, selected models, available proposal records, and classical/random controls. The independent-reference WENO checks are under `experiments/review_gates/`; only the controls relevant to this manuscript and their dependencies are included. Native execution and eager-reference validation sources and records accompany `ndvm/` and `neural_compiler/`.

Choose the runtime deliberately. The root `ndvm/` preserves the historical implementation used for the recorded searches and timing measurements. `runtime_corrected/` supplies the later numerical repairs as a separate source variant, with build and regression instructions. A reproduction must state which variant and precision it uses; changing them can change optimization trajectories. The fixed-program float64 WENO transfer evaluator is implemented in PyTorch and does not require the native extension. Environment records distinguish the original Linux execution setting from the environment used for submission-package checks.

Reproduction limits. The original live model-serving revisions and all discarded search responses are not available. Model names and decoding settings identify the requested services, not immutable model weights. The supplied historical logs may contain coarse printed precision, and archived proposal pools omit records lost before or between checkpoints. Neural training code and logged evaluations are provided; original trained weights were not saved, so reproducing those controls requires retraining rather than loading a checkpoint. These limitations are recorded alongside the

relevant commands. The locally retained historical Kalman analysis predates the Qwen arms; the package includes their selected programs, the recorded aggregate table, and a separately identified submission-time recheck that reproduces all eight Qwen-arm recovery cells.