GROW OR PRUNE? CONSTRUCTIVE NEUROEVOLUTION OF KOLMOGOROV–ARNOLD NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Kolmogorov–Arnold networks (KANs) are usually trained as fixed, over-parameterized graphs, then pruned and converted to equations. We study the constructive alternative: grow the graph from a minimal network. Constructive KAN (C-KAN) evolves typed, spline-edge graphs with short Lamarckian adaptation. Identity-oriented growth, fan-in normalization, amplitude-corrected Safe Mutation and behavioral speciation address mismatches between NEAT and function-valued edges; several species supply candidates for symbolic conversion. On 30 clean SRSD equations, C-KAN-GSR returns 12 high-accuracy equations, compared with none for two fixed-template PyKAN pipelines, and its delivered equations are more often valid and smaller. It also improves black-box SRBench predictive AUC over both templates. Yet neither KAN pipeline recovers an exact skeleton: PySR and KAN-SR recover 43.3% and 15.8%, respectively. A post-hoc constant refit raises PyKAN-1 above C-KAN in high-accuracy rate, but C-KAN's expressions remain smaller and structurally closer to the targets. Stage records locate C-KAN's main bottleneck in graph search: 17 of 30 equations lack an accurate prepared network; conversion and export cause further losses. Small ablations associate the KAN-specific mechanisms with higher validity and lower measured cost, without establishing a recovery gain. Constructive growth makes sparse KANs easier to export, but graph discovery alone does not solve equation recovery.

1 INTRODUCTION

Kolmogorov–Arnold networks (KANs) place learnable univariate functions on the edges of a graph, so a trained network can be read, edge by edge, as a composition of one-dimensional functions (Liu et al., 2025). This makes them an attractive intermediate representation for symbolic regression: gradient descent shapes the functions, and symbolic primitives are assigned afterwards. Yet the graph itself is fixed in advance. Depth, width and, in KAN 2.0, the placement of multiplications are chosen before training (Liu et al., 2024). The standard recipe is *destructive*: train an over-parameterized template, prune it, then convert what survives. Pruning can remove structure, but it cannot recover a composition the template excluded, and each trained network yields a single extraction attempt.

We study the *constructive* alternative: grow the KAN graph from a minimal network so that structure is added only when the data reward it. NEAT offers the machinery, with historical gene alignment, speciation and complexification from minimal genomes (Stanley & Miikkulainen, 2002). It does not transfer to spline edges unchanged. Growth pushes downstream inputs off their spline grids, coefficient mutations reshape functions unpredictably, coefficient distances do not measure functional difference, multiplication is hard to assemble from additive edges, and a single extracted network leaves no fallback when symbolic conversion fails (Section 3). **Constructive KAN** (C-KAN) addresses each failure while keeping the KAN edge representation and its per-edge symbolic conversion unchanged. Its mechanisms are identity-constructed growth, gradual fan-in normalization, smoothed geometric multiplication, amplitude-corrected Safe Mutation, bounded behavioral speciation and recovery from several species.

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.

We use symbolic regression on SRBench and SRSD-Feynman to test what growing, rather than pruning, changes. Our central finding is that growing and pruning fail at opposite stages. Stage-level telemetry shows that the dense template fits more equations at first but loses them through pruning and a poorly fitted export, whereas the grown graph fits fewer but keeps the accuracy it finds. As delivered, C-KAN is therefore more accurate and more often valid than fixed-template PyKAN on the completed clean benchmarks, and it returns much smaller equations. A post-hoc refit of PyKAN's exported constants lets it match or exceed C-KAN's accuracy, so what persists is structural: C-KAN's equations stay smaller and closer to the target.

The graph alone, however, is not enough for equation recovery. C-KAN recovers no exact skeleton, whereas PySR, which searches expressions directly, recovers 43%, and KAN-SR, which wraps KANs in symbolic decomposition and search, recovers 16%. Most of C-KAN's losses occur in the search itself; the rest occur at export, where geometric nodes and affine-wrapped primitives yield forms that match the target numerically but not structurally. Small descriptive ablations associate Safe Mutation and behavioral speciation with higher validity and lower measured cost, without a detected accuracy gain.

Our contributions are:

- **A constructive search space for KANs.** Typed sparse DAGs, per-edge spline resolution and post-adaptation selection, with a template–realization–parameter decomposition $g = (c, \xi, \theta)$ (Section 3).
- **An account of what breaks when NEAT meets spline edges, with targeted mechanisms.** This includes a fan-in-independent activation bound, change-at-application bounds for growth, and closure of crossover within the fitter parent's template.
- **A traceable recovery pipeline.** It extracts equations from several species of the evolved population and records every stage, so that failures can be attributed to the search, the conversion or the export.
- **An empirical comparison on two clean benchmark tracks.** It reports stage-level accuracy, failures, a post-hoc clean-up sensitivity analysis and ablations (Section 4).

2 Related Work

Symbolic regression and KANs. Genetic programming searches expression trees, and Operon and PySR add numerical constant optimization (Koza, 1992; Burlacu et al., 2020; Cranmer, 2023). Neural methods propose expressions with learned policies or transformers (Petersen et al., 2021; Kamienny et al., 2022; Shojaee et al., 2023). KANs instead fit a network of univariate functions and convert it afterwards (Liu et al., 2025; 2024). The conversion backends include per-edge fitting with a linear-first rule (Knottenbelt et al., 2024) and greedy in-context symbolic regression (Sovrano et al., 2026). KAN-SR guides separability and symmetry decomposition with a trained KAN (Bühler & Guillén-Gosálbez, 2025). All of these KAN approaches fix the architecture before training; C-KAN searches it.

Neuroevolution and constructive growth. NEAT introduced historical alignment, speciation and complexification from minimal networks (Stanley & Miikkulainen, 2002). Prior work applies NEAT to KAN structure for control (Nguyen et al., 2025), and we build on this precedent for symbolic recovery. Non-evolutionary constructive methods also grow networks: cascade-correlation, Net2Net, network morphism and GradMax (Fahlman & Lebiere, 1990; Chen et al., 2016; Wei et al., 2016; Evci et al., 2022). C-KAN shares their aim of reusing learned behavior, but its spline transfers are approximate.

Hybrid search and diversity. Lamarckian evolution with gradient refinement appears in ESGD, PBT and LEMONADE (Cui et al., 2018; Jaderberg et al., 2017; Elsken et al., 2019). Safe Mutation scales perturbations by output gradients (Lehman et al., 2018), and C-KAN corrects its spline-amplitude coupling. Behavioral comparison has precedents in semantic GP and novelty search (Moraglio et al., 2012; Lehman & Stanley, 2011); C-KAN uses it only for speciation. Appendix K gives an extended discussion, and Table 6 compares the searched objects.

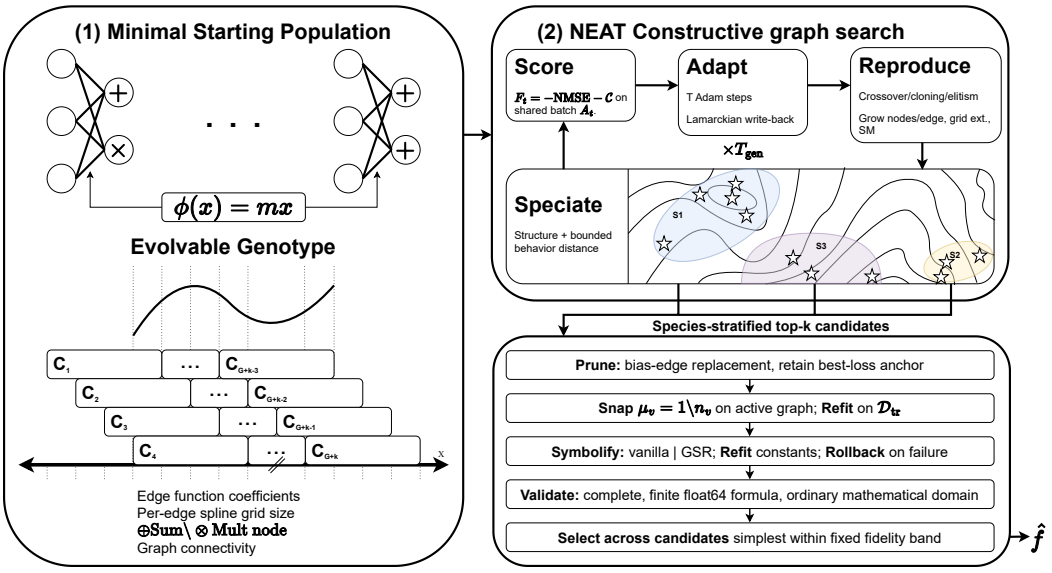

Figure 1: C-KAN. A minimal population evolves sparse spline-edge graphs through reproduction, short Lamarckian adaptation, scoring and structural–behavioral speciation. Species-stratified candidates are then pruned by mean replacement, converted to symbols by vanilla or GSR extraction, and validated before the simplest equation within a fidelity band is returned. The colored regions illustrate species.

3 METHOD: CONSTRUCTIVE KAN

C-KAN searches the part of an equation that a train–prune–symbolify pipeline must guess: the computation graph. Evolution is the outer loop over graphs and parameter starting states, and gradient descent is the inner loop over edge functions (Figure 1). The full algorithm, the provenance of each component and the proofs are in Appendices A and D–H.

Why NEAT does not transfer to KANs unchanged. Replacing scalar weights by splines creates five problems:

- **(F1) Domain–range misalignment.** A spline acts only on its knot grid, so growth that changes a node's fan-in or activation distribution can push downstream inputs off their grids.
- **(F2) Mutation severity.** A gene is now a function, and an isotropic coefficient perturbation can reshape it by orders of magnitude more on one edge than another.
- **(F3) Uninformative compatibility.** Coefficients on different grids are not comparable, so NEAT's weight distance does not measure functional difference.
- **(F4) Indirect multiplication.** An additive KAN forms $x_1 x_2$ only through coordinated multi-edge identities.
- **(F5) Extraction fragility.** An accurate network can fail to symbolify, and a single final candidate leaves no fallback.

Each C-KAN component targets one or more of these problems. Table 4 marks which components are new, adapted or borrowed.

3.1 SEARCH SPACE

An individual is $g = (c, \xi, \theta)$. The *template* c is a typed acyclic graph of historical NEAT genes, with node types $\alpha_v \in \{\oplus, \otimes\}$ fixed at birth. The *realization* ξ fixes the enabled genes, each edge's knot grid with resolution G_e, and each node's scale state μ_v. The parameters θ are the spline coefficients

and edge scales. Following KAN (Liu et al., 2025), edge $e : u \to v$ computes

$$z_e = \phi_e(h_u), \qquad \phi_e(a) = w_{b,e}\,\mathrm{SiLU}(a) + w_{s,e}\sum_j c_{e,j} B_{j,k}(a; t_e), \tag{1}$$

with cubic B-splines. With n_v active inputs and $p_v = \prod_{e\in\mathrm{In}(v)} z_e$, a node computes

$$h_v = \mu_v \textstyle\sum_{e\in\mathrm{In}(v)} z_e \;\; (\alpha_v = \oplus), \qquad h_v = p_v\big(p_v^2 + \epsilon_\times\big)^{(\mu_v-1)/2} \;\; (\alpha_v = \otimes). \tag{2}$$

One scale state serves both node types. At additive nodes it is a mean multiplier, and at multiplicative nodes a root exponent. For multiplicative nodes, Eq. (2) is a smooth, sign-preserving surrogate of $\mathrm{sign}(p_v)|p_v|^{\mu_v}$, which approximates a signed geometric mean at $\mu_v = 1/n_v$. Topology, resolution and node types all evolve. Raw products let activations explode as fan-in grows (F1, F4); the geometric form prevents this:

Proposition 1 (Fan-in-independent activation scale)**.** *If every incoming $|z_e| \le M$ and $\mu_v = 1/n_v$, then $|h_v| \le M$ for both node types and every fan-in. The raw product ($\mu_v = 1$) attains M^{n_v}.*

3.2 Selecting structures after adaptation

Each generation, every non-elite individual takes $T = 5$ Adam steps, $\theta^+ = U_T(g; A_t)$, on a mini-batch A_t shared by the population. It is then scored and θ^+ is written back (Lamarckian inheritance):

$$F_t(g) = -\mathcal{L}_{A_t}(f_{c,\xi,\theta^+})/s_y^2 - \lambda_E|E_{\mathrm{act}}| - \lambda_G \textstyle\sum_{e\in E_{\mathrm{act}}} G_e. \tag{3}$$

Here s_y^2 is the training-target variance. Parsimony therefore acts on selection, not on the gradient, and selection judges how well a structure *adapts* rather than its unadapted coefficients. A Jacobian argument and the family formalism $\mathcal{M}_c$ are in Appendix A.

3.3 Variation operators for function-valued genes

Growth at the identity (F1). With $w_b = w_s = 1$, the edge

$$\phi^{[r]} = \mathrm{SiLU} + \widehat{\Pi}_t\big[r - \mathrm{SiLU}\big] \tag{4}$$

realizes a target function r, up to a least-squares projection error. *Add-node* splits $u \to v$ into $u \to w \to v$ with $\phi_{u\to w} = \phi^{[\mathrm{id}]}$ and copies the old edge onto $w \to v$. *Add-connection* into a multiplicative node enters at $\phi^{[1]}$ plus jitter. Into an additive node, it enters with a random base weight and a near-silent spline. Proposition 2 (Appendix F) bounds the change each operator causes at application. Grid extension (a heritable resolution mutation) and periodic grid adaptation keep each edge's grid on its inputs.

Gradual normalization (F1). Snapping μ_v to $1/n_v$ after a new edge would rescale all existing contributions at once, so the scale drifts instead:

$$\mu_v \leftarrow \beta\mu_v + (1-\beta)/\widehat{n}_v, \qquad \beta = 0.5. \tag{5}$$

Crossover within the fitter parent's template (F2). Matching genes inherit a whole spline payload from one parent, and unmatched genes come from the fitter parent. The child therefore has exactly the fitter parent's template and searches its behavior family without mixing coefficients across grids (Appendix F).

3.4 Stabilizing and diversifying functional evolution

Amplitude-corrected Safe Mutation (F2). Safe Mutation scales each perturbation by the output's sensitivity to that parameter (Lehman et al., 2018). For a spline coefficient, that sensitivity is gated by the edge amplitude:

$$\partial f/\partial c_{e,j} = (\partial f/\partial z_e)\, w_{s,e}\, B_{j,k}(h_u; t_e). \tag{6}$$

An edge with a quiet spline branch therefore looks insensitive. It hits the floor $\sigma_{\min}$ and receives steps up to $1/\sigma_{\min}$ times larger, which surface once the amplitude grows. C-KAN divides the amplitude out before the floor, for spline coefficients only:

$$\widetilde{\sigma}_{e,j} = \sigma_{e,j}/\max(|w_{s,e}|, 10^{-12}), \qquad \Delta\theta_j = \zeta_j/\max(\widetilde{\sigma}_j, \sigma_{\min}). \tag{7}$$

Bounded behavioral speciation (F3). C-KAN replaces NEAT's coefficient distance with a bounded comparison of predictions on a shared probe B_t:

$$\delta(g,h) = \frac{c_E E + c_D D}{N} + c_B\big[1 - e^{-u_{B_t}(g,h)}\big], \qquad u_B = \frac{\|\mathbf{b}_B(g) - \mathbf{b}_B(h)\|_2^2}{|B|\, s_y^2}. \tag{8}$$

Here E and D count excess and disjoint genes. The bound keeps one outlying individual from fragmenting the population.

3.5 Population-backed equation recovery

Recovery works on several candidates rather than one (F5):

- *Candidates.* C-KAN takes K champions stratified across species.
- *Pruning.* Edges are tried in KAN attribution order. Each trial replaces an edge by a constant edge carrying its mean output. This keeps the receiving fan-in and operating point, where deletion would change both. A trial is accepted only if

$$\mathcal{L}_{\mathcal{D}_{\text{sel}}}(f_{\text{trial}}) \le (1+\tau_p) L_{\min} + \alpha_p, \tag{9}$$

  where $L_{\min}$ is the best holdout loss so far. Tolerated degradation therefore cannot compound.
- *Conversion.* Each pruned candidate is converted by per-edge fitting (vanilla) or by greedy in-context symbolic regression (GSR) (Sovrano et al., 2026), refit, and checked as an ordinary float64 formula.
- *Selection.* Among the valid equations $\mathcal{V}$, with holdout losses ℓ_s and best loss ℓ_*, C-KAN returns

$$s^* = \operatorname*{lex\,argmin}_{s\in\mathcal{V}:\ \ell_s \le (1+\tau_e)\ell_* + \alpha_e} \big(\text{size}(s),\ \ell_s\big). \tag{10}$$

Every stage is logged, so each failure can be traced to the search, the conversion or the export.

4 Experiments

4.1 Setup

We use symbolic regression as a test of whether a grown KAN is a better intermediate representation than a pruned one. *Track A* measures held-out prediction on 6 black-box (BB) and 6 first-principles (FP) SRBench datasets (Aldeia et al., 2025), with ten seeds. *Track B* measures equation recovery on 30 clean SRSD-Feynman equations (10 Easy, 10 Medium, 10 Hard), with five seeds (Matsubara et al., 2024). We also collected a corrupted SRSD panel, but its input hashes differ across machines; it is excluded from controlled comparisons and reported descriptively in Appendix Q. For Track B, one seed per method and equation is chosen by the official validation loss. A skeleton counts as recovered when its normalized tree edit distance (NED) to the ground truth is zero. Failed runs score NED 1, and overall rates weight the difficulty groups as in SRSD.

The destructive baseline is PyKAN (Liu et al., 2025) with a fixed one- or two-hidden-layer template (PyKAN-1, PyKAN-2), trained, pruned and converted per edge. We also compare against KAN-SR (Bühler & Guillén-Gosálbez, 2025), PySR (Cranmer, 2023), Operon (Burlacu et al., 2020) and TPSR (Shojaee et al., 2023). C-KAN runs 100 generations of 150 individuals within one hour on one CPU thread, with $K = 5$ candidates. Its GSR and vanilla equations come from the same evolution, and GSR is extracted first.

Only PyKAN's vanilla-extraction runs were completed, and they use PyKAN's default sparsity weight $\lambda = 0$. The comparison roster was revised after seeing results, so all tests are exploratory, and most PySR, Operon and TPSR Track A scores come from published SRBench records. Appendices L and N give the full protocol, per-track analyses, intervals and tests.

Table 1: Main results. Track A: black-box AUC of the run-level clipped-R^2 profile, median first-principles test R^2, and valid runs. Track B: validation-selected skeleton recovery (SR), mean scored NED, test $R^2 > 0.999$ rate, and fully evaluable runs; rates are difficulty-weighted. †Mostly published SRBench records with provisional validity. KAN-SR was not run on Track A.

	Track A (SRBench)			Track B (clean SRSD)			
Method	BB AUC ↑	FP R^2 ↑	Valid	SR ↑	NED ↓	R^2>.999	Valid
C-KAN-GSR	0.332	0.922	86.7	0.0	0.957	37.5	50.3
C-KAN-Vanilla	0.276	0.887	86.7	0.0	0.974	25.0	37.0
PyKAN-1 (destructive)	0.249	−2.03	80.0	0.0	1.000	0.0	37.3
PyKAN-2 (destructive)	0.149	−11.46	73.3	0.0	1.000	0.0	1.3
KAN-SR	—	—	—	15.8	0.567	77.5	91.3
PySR	0.435	0.984	100†	43.3	0.426	71.7	96.7
Operon	0.435	0.936	100†	0.0	0.808	55.0	62.7
TPSR	0.179	0.956	100†	0.0	0.975	0.0	85.0

Table 2: Clean Track B equations (of 30, best of five seeds) on which a model is accurate after each pipeline stage. *Accurate* means NMSE below 10^{-3} on the method's internal training-only holdout; the last column is test $R^2 > 0.999$ of the validation-selected equation. C-KAN prunes and refits in one step; PyKAN does not log its symbolic model before refitting.

Method	Trained network	After pruning	After refit / conversion	Returned equation
C-KAN-GSR	9	13 → 12		12
PyKAN-1	22	5	17	0

4.2 Constructive against destructive KANs

As delivered, C-KAN-GSR is more accurate and more often valid than both fixed templates on the clean benchmarks (Table 1). On Track A, its black-box AUC is 0.332, against 0.249 and 0.149 for PyKAN-1 and PyKAN-2; its median first-principles R^2 is 0.922, whereas both PyKAN medians are negative; and it returns valid equations in 86.7% of runs, against 80.0% and 73.3%. On Track B, C-KAN-GSR fits 12 of 30 equations to test $R^2 > 0.999$ and C-KAN-Vanilla fits 8, while neither PyKAN template fits any. C-KAN's equations are also much smaller, with a median of 30 nodes on Track A against 157 and 1,178 for the two templates. When both sides use vanilla per-edge conversion, C-KAN still leads on every accuracy measure, and Track B validity is similar (37.0% against 37.3%).

None of the four KAN pipelines recovers an exact skeleton, and the exploratory structure-score tests between C-KAN and PyKAN are not significant after Holm correction, because both sit near the NED ceiling.

Table 2 shows where each approach loses accuracy, and the two pipelines fail at opposite ends. The dense template fits more equations than C-KAN's search does (22 against 9), but pruning leaves 5; refitting restores 17, yet none of the returned equations is accurate. The grown graph fits fewer equations but keeps them: pruning and refitting raise 9 to 13, conversion keeps 12, and the returned equations stay accurate.

Much of PyKAN's final loss, however, lies in the constants of its exported equations rather than in their form. After seeing the results, we refit the constants of every returned Track B equation by least squares on the training data (Section 4.3, Appendix R). This raises PyKAN-1's high-accuracy rate from 0 to 54%, above C-KAN-GSR's 45% under the same treatment. The median test R^2 of its re-scored equations rises from 0.78 to 0.9998. After term elimination, both recover 5.8% of skeletons. C-KAN's equations remain smaller and structurally closer to the target: median size ratio 2.0 against 7.4, and mean NED 0.873 against 0.942.

The constructive pipeline's accuracy advantage therefore comes largely from keeping and refitting what it finds. Its lasting advantage is structural. Two limits bound this reading. First, PyKAN was trained without the sparsity penalty ($\lambda = 0$) that the KAN recipe uses before pruning, so a

regularized template may prune more gracefully. Second, PyKAN's post-pruning refit diverged in 85 of 150 Track B runs; we did not test whether regularization would prevent this. We therefore read the comparison as locating where each pipeline is fragile, not as a fixed accuracy margin.

4.3 Where constructive KANs still fall short

The symbolic baselines remain far ahead on recovery. PySR recovers 43.3% of the clean skeletons and KAN-SR 15.8%, against C-KAN's 0% (PySR versus C-KAN: Holm-adjusted exact McNemar $p < 0.001$). PySR and Operon also lead on black-box AUC (0.435 each) and produce smaller equations.

C-KAN changes only the KAN's graph: its edges are standard splines and its conversion is the standard per-edge one. KAN-SR instead wraps standard single-layer KANs in symbolic machinery: separability and symmetry decomposition, brute-force search for small inputs, and output transforms and rounding. PySR searches expressions directly.

Appendix Table 21 partitions the 30 clean equations: search accounts for 17 losses, conversion for 3, and export or structural mismatch for 7; another 3 are recovered only after post-hoc clean-up. Export failures include tiny residual terms (`ii.8.31`, with test $R^2 = 1.000$ but NED 1), sign-dependent geometric-node forms (`i.12.1`) and aliased primitives such as $\sin(x_2 - 3\pi/2)$ in place of $-\cos x_2$ (`ii.15.4`).

We also tested whether ordinary symbolic post-processing closes the gap. After seeing the results, we applied KAN-SR's own rounding, refit and term elimination to every Track B method's equations and re-scored them with the benchmark code (Appendix R). C-KAN and PyKAN-1 each gain two equations (0 to 5.8%). Operon rises from 0 to 24.2%, KAN-SR from 15.8% to 30.8%, and PySR falls from 43.3% to 27.5%. The benchmark's canonicalizer removes a constant only when it equals one exactly, so the rounding and refit can break PySR's constant-free exact matches. Recovery on SRSD is thus sensitive to post-processing for every method. The KAN pipelines' gap, however, is structural rather than cosmetic.

4.4 Ablations

We study Safe Mutation (SM) and behavior-distance speciation in a 2×2 design on 12 clean Track B equations (3 Easy, 4 Medium, 5 Hard). A second, six-equation subset (1 Medium, 5 Hard) tests candidate count K, gradient steps and grid extension. The candidate-count pilots differ in source version and realized time; all comparisons are descriptive (Table 3; Appendix S).

No variant recovers a skeleton. In the component study, the full configuration produces fully evaluable equations in 28/60 runs, against 19–24/60 when either mechanism is removed, and has 23 canonicalization timeouts against 29–33. Removing SM raises the median measured fit time from 3,240 to 4,152 seconds on matched hardware. Accuracy differs by at most one equation. Species-count trajectories are similar, though counts alone do not measure behavioral diversity. These pilot observations suggest a validity and cost benefit without an established accuracy gain.

Figure 2 shows that the numerically best network supplies the returned equation in only 11–24% of completed candidate-count runs; the simplicity tie-break changes just 11 of 81 reproduced selections. Conversion often reorders the numerical ranking, though these counts do not establish a gain over champion-only extraction. On the six-equation subset, removing gradient steps lowers the median fit time from 3,059 to 1,262 seconds without a detectable accuracy loss at the zero-accuracy floor; removing grid extension lowers validity from 10/30 to 6/30 runs.

Table 3: C-KAN GSR ablations on Track B. The four component rows share equations and nominal settings. The six-equation rows share equations and seeds, but the $K = 10$ pilot had a longer nominal cap and older pilots could overrun; interpret cross-batch differences descriptively. n_d gives complete datasets. SR and accuracy use validation-selected equations; NED assigns 1 to failures. Nodes is the median size among fully evaluable selected equations; validity is over observed runs. Hardware differs across some variants, so runtime is omitted.

Variant	n_d	SR (%)	Acc. (%)	NED ↓	Nodes	Valid (%)
Component study: 12 equations (3 Easy, 4 Medium, 5 Hard), $K = 10$, 5000 s						
Full, $K = 10$	12/12	0.0%	16.7%	0.970	44.0	46.7%
−SM	12/12	0.0%	16.7%	0.990	19.0	31.7%
−behavior	12/12	0.0%	16.7%	1.000	30.0	40.0%
−SM − behavior	12/12	0.0%	25.0%	0.955	22.0	31.7%
Single changes: 6 equations (1 Medium, 5 Hard), 3600 s unless noted						
Main, $K = 5$ (K-study batch)	6/6	0.0%	0.0%	0.951	19.0	30.0%
Full, $K = 7$	6/6	0.0%	0.0%	0.954	44.0	30.0%
Full, $K = 10$ (5000 s)	6/6	0.0%	0.0%	0.954	35.0	36.7%
Main, $K = 5$ (main batch)	6/6	0.0%	0.0%	0.951	19.0	33.3%
−gradient steps	6/6	0.0%	0.0%	0.942	18.0	46.7%
−grid extension	6/6	0.0%	0.0%	0.954	10.0	20.0%

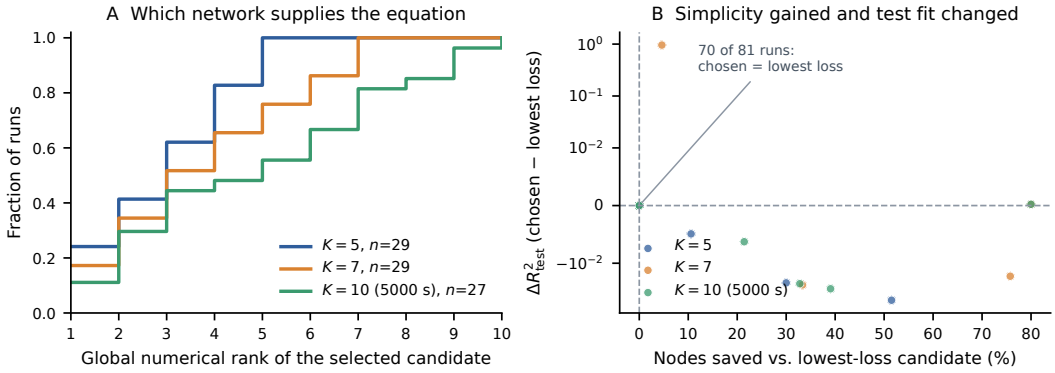

Figure 2: Six-equation candidate-count pilots (GSR). (A) Numerical rank of the network supplying the selected equation; species stratification can select ranks above K. (B) Expression nodes saved by the selected equation versus the lowest-validation-loss symbolic candidate, and the corresponding change in held-out R^2. Test data were used only after selection. Nominal caps and realized runtimes differ across pilots.

5 Conclusion

Growing a KAN's graph instead of pruning a template required mechanisms that scalar-weight neuroevolution does not need. On the completed clean benchmarks, the constructive search yields equations that, as delivered, are more accurate, more often valid and much smaller than those of the fixed-template PyKAN baselines. The grown networks keep their accuracy through pruning and conversion. The template loses much of its accuracy there, though a post-hoc refit of its constants recovers it. What persists after equal post-processing is structural: C-KAN's equations are smaller and closer to the target.

Recovering equations needs more than the graph. C-KAN keeps KAN's spline edges and per-edge conversion, and it recovers no exact skeleton. Methods that add symbolic structure around the network (KAN-SR) or search expressions directly (PySR) recover far more. Our stage-level records point to two targets. The first is a more reliable and efficient search, since the current one finds no accurate prepared network for 17 of 30 SRSD equations and fails on the sole 50-input dataset. The second is export-aware design, such as clean power forms under known input signs, primitive de-aliasing and pricing expression size rather than graph size.

The graph-level search and population-backed recovery offer a design pattern for other KAN edge families, but transfer would require edge-specific mutation, adaptation and symbolic export rules that we have not tested.

Limitations. The clean SRSD panel uses a curated 30 equations and five seeds; the ablations use 12 and 6 equations where recovery is zero throughout. The destructive PyKAN baseline uses the default $\lambda = 0$, and the comparison roster was revised after seeing results, so a tuned template may narrow the gap. Track C corrupted-array hashes disagree across machines and cannot support controlled cross-method or clean-to-corrupted effects. Heterogeneous machines and older pilot budget overruns limit runtime comparisons. The stage-level and clean-up analyses were post hoc.

AI USE STATEMENT

Generative AI tools assisted with manuscript drafting and revision, mathematical exposition and consistency checks, and implementation and debugging of the accompanying software. The authors are responsible for the final claims, citations, code and empirical evaluation.

REPRODUCIBILITY STATEMENT

Algorithm 1 and Appendix C specify execution order and budget accounting. Appendices D–H provide initialization rules, transfer assumptions, population management, pruning and conversion checks; Appendix J gives the main `ckan_k5` settings.

The benchmark code supports the full dataset and seed rosters as well as the smaller main-paper profiles: 12 SRBench datasets with the first 10 official seeds for Track A, and the same 30 SRSD equations with the first 5 seeds for Tracks B and C. Track C is designed to use one fixed corruption realization per equation across methods; the current collected hashes disagree across machines, so these outcomes are reported provisionally pending reconciliation. Internal fitting and candidate selection use training-derived splits; official validation and test labels remain external. Configuration hashes, environments, attempt identifiers and failure records support the selection of compatible runs for analysis. Older pilot budgets and settings are reported separately in Appendix S. The post-hoc clean-up and stage-level diagnostics of Appendix R read the result ledgers without modifying them. Their scripts are included.

REFERENCES

Guilherme S. Imai Aldeia, Hengzhe Zhang, Geoffrey Bomarito, Miles Cranmer, Alcides Fonseca, Bogdan Burlacu, William G. La Cava, and Fabrício Olivetti de França. Call for action: Towards the next generation of symbolic regression benchmark. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (Symbolic Regression Workshop)*, 2025. URL https://arxiv.org/abs/2505.03977.

Mohammed Ghaith Altarabichi. Rethinking the function of neurons in KANs. *arXiv preprint arXiv:2407.20667*, 2024. URL https://arxiv.org/abs/2407.20667.

Marco Andrea Bühler and Gonzalo Guillén-Gosálbez. KAN-SR: A Kolmogorov-Arnold network guided symbolic regression framework. *arXiv preprint arXiv:2509.10089*, 2025. URL https://arxiv.org/abs/2509.10089.

Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. Operon C++: An efficient genetic programming framework for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1562–1570, 2020.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2Net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.05641.

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

Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. In *Advances in Neural Information Processing Systems*, 2020.

Xiaodong Cui, Wei Zhang, Zoltán Tüske, and Michael Picheny. Evolutionary stochastic gradient descent for optimization of deep neural networks. In *Advances in Neural Information Processing Systems*, 2018. URL https://arxiv.org/abs/1810.06773.

Carl de Boor. *A Practical Guide to Splines*. Springer, 1978.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via Lamarckian evolution. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1804.09081.

Utku Evci, Bart van Merriënboer, Thomas Unterthiner, Max Vladymyrov, and Fabian Pedregosa. GradMax: Growing neural networks using gradient information. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2201.05125.

Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pp. 524–532, 1990.

Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. *Complex Systems*, 1 (3):495–502, 1987.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017. URL https://arxiv.org/abs/1711.09846.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. In *Advances in Neural Information Processing Systems*, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

William Knottenbelt, Zeyu Gao, Rebecca Wray, Woody Zhidong Zhang, Jiashuai Liu, and Mireia Crispin-Ortuzar. CoxKAN: Kolmogorov–arnold networks for interpretable, high-performance survival analysis. *arXiv preprint arXiv:2409.04290*, 2024. URL https://arxiv.org/abs/2409.04290.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley. Safe mutations for deep and recurrent neural networks through output gradients. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018. URL https://arxiv.org/abs/1712.06563.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a. URL https://arxiv.org/abs/1806.09055.

Qiang Liu, Lemeng Wu, and Dilin Wang. Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems*, 2019b. URL https://arxiv.org/abs/1910.02366.

Ziming Liu, Pingchuan Ma, Yixuan Wang, Wojciech Matusik, and Max Tegmark. KAN 2.0: Kolmogorov–arnold networks meet science. *arXiv preprint arXiv:2408.10205*, 2024. URL https://arxiv.org/abs/2408.10205.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. KAN: Kolmogorov–arnold networks. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2404.19756.

Yoshitomo Matsubara, Naoya Chiba, Ryo Igarashi, and Yoshitaka Ushiku. Rethinking symbolic regression datasets and benchmarks for scientific discovery. *Journal of Data-centric Machine Learning Research*, 2024. URL https://arxiv.org/abs/2206.10540.

Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. Geometric semantic genetic programming. In *Parallel Problem Solving from Nature – PPSN XII*, pp. 21–31, 2012.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015. URL https://arxiv.org/abs/1504.04909.

Phong Huy Nguyen, Xuan Phuc Nguyen, and Ngoc Hoang Luong. Optimization of Kolmogorov-Arnold networks for reinforcement learning via neuroevolution of augmenting topologies. In *14th International Symposium on Information and Communication Technology (SOICT), Program Book*, 2025. URL https://soict.org/wp-content/uploads/2025/12/SOICT2025-ProgramBook.pdf. Conference abstract.

Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Claudio P. Santiago, Soo K. Kim, and Joanne T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. Transformer-based planning for symbolic regression. In *Advances in Neural Information Processing Systems*, 2023.

Francesco Sovrano, Lidia Losavio, Giulia Vilone, and Marc Langheinrich. In-context symbolic regression for robustness-improved kolmogorov–arnold networks. *arXiv preprint arXiv:2603.15250*, 2026. URL https://arxiv.org/abs/2603.15250.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. doi: 10.1162/106365602320169811.

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

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016. URL https://arxiv.org/abs/1603.01670.

Darrell Whitley, V. Scott Gordon, and Keith Mathias. Lamarckian evolution, the Baldwin effect and function optimization. In *Parallel Problem Solving from Nature – PPSN III*, pp. 5–15, 1994.

A EXTENDED METHOD DESCRIPTION

This appendix collects the parts of the method description that the main text summarizes: component provenance, the behavior-family formalism, the complete algorithm, the operator taxonomy, and longer discussions of scale control, crossover, Safe Mutation, speciation and recovery. Equations referenced here are numbered in Section 3.

A.1 COMPONENT ROLES AND PROVENANCE

Table 4: Component roles and provenance. The design combines KAN-specific mechanisms with established search and extraction components.

Component	Targets	Relation to prior work
Typed DAG, evolvable edge resolution	F1, F4	C-KAN search space; NEAT and KAN ingredients
Smoothed geometric aggregation; scale drift	F1, F4	C-KAN design; mean aggregation adapted
Post-adaptation selection and inheritance	F1, F2	Lamarckian / memetic search
Fitted identity insertion; grid transfer	F1	NEAT growth adapted to spline edges
Whole-edge, template-preserving crossover	F2	NEAT inheritance, with typed-DAG handling
Amplitude-corrected Safe Mutation	F2	C-KAN correction to output-gradient SM
Bounded behavioral compatibility	F3	C-KAN formulation of behavior-based speciation
Species-stratified candidate recovery	F5	C-KAN use of the evolved population
Mean replacement and best-loss pruning	F5	C-KAN pruning rule; KAN attribution order
Per-edge / GSR conversion and fidelity band	F5	Borrowed backends, adapted fitting and selection

A.2 SCALE CONTROL

Summation can increase magnitudes with fan-in, shifting the operating points of outgoing splines. Mean aggregation mitigates this scaling effect (Altarabichi, 2024). At fixed topology its multiplier can be absorbed into edge functions; this does not guarantee that activations fall inside an arbitrary downstream grid. Products compound the problem: n_v factors of magnitude M reach M^{n_v}. Smoothed geometric aggregation controls the final activation scale (Proposition 1).

In preliminary product-node experiments, raw products more often produced exploding or nonfinite activations; geometric aggregation was substantially more stable. The bound explains how normalizing each node output limits growth across successive layers, while leaving a narrower possibility of intermediate floating-point overflow within a node (Appendix F). KAN 2.0 places multiplication nodes by hand in a fixed architecture (Liu et al., 2024). Here their placement and their fan-in both evolve, which is why their scale must be controlled. The root exponent cannot be absorbed by neighbouring affine maps, so it is exported verbatim. Section N.2 shows that this exported form can hide a correct product from structural scoring.

A.3 BEHAVIOR FAMILIES, THE SEARCH OBJECTIVE AND THE ALGORITHM

On reference inputs B, write $\mathbf{b}_B(g) = (f_g(x))_{x\in B}$. A template and a realization define the behaviors they can reach,

$$\mathcal{M}_{c,\xi}(B) = \{\mathbf{b}_B(c,\xi,\theta) : \theta \in \Theta_{c,\xi},\ \mathbf{b}_B \text{ finite}\}, \qquad \mathcal{M}_c(B) = \textstyle\bigcup_{\xi\in\Xi_c} \mathcal{M}_{c,\xi}(B), \tag{11}$$

where reachable sets are taken on a finite sample and are not assumed to be manifolds. Parameter moves stay in $\mathcal{M}_{c,\xi}$, realization moves in $\mathcal{M}_c$, and only structural mutation changes c (Table 5). Let $U_T(g; A)$ return the parameters after T Adam steps (Kingma & Ba, 2015) to θ on a sample A. Idealized, C-KAN seeks structures and *starting states* that adapt well,

$$\min_{c,\ \xi,\ \theta_0} \ \mathbb{E}_A\Big[\mathcal{L}_A\big(f_{c,\xi,U_T((c,\xi,\theta_0);A)}\big)\Big]/s_y^2 + \mathcal{C}(c,\xi). \tag{12}$$

Algorithm 1 C-KAN: constructive search and symbolic recovery

Require: $\mathcal{D}_{\text{fit}}, \mathcal{D}_{\text{sel}}$; population P; generation cap H; local steps T; candidates K; shared fit deadline

```
Initialize P genomes with direct input–output edges and no hidden nodes
for t = 1, ..., H while fit time remains do
    if t > 1 then
        Reproduce using previous fitnesses and species (Algorithm 2)
        Drift non-elite node scales toward reciprocal enabled fan-in
    end if
    Draw shared minibatch A_t; set θ_g ← U_T(g; A_t) for non-elites
    Score all individuals by Eq. (3); refresh buffers from D_fit
    Speciate on a shared probe; update stagnation and population statistics
    Every T_grid generations, adapt non-elite grids using buffered inputs
end for
Select up to K candidates by species and D_sel loss
Prepare candidates: mean-compensated pruning, active-fan-in snap, numerical refit
for backend in (GSR, vanilla), while fit time remains do
    Convert separate prepared copies; refit and validate each completed equation
    Emit the backend's Eq. (10) selection, or its recorded failure
end for
```

Selection maximizes the one-sample estimate of Eq. (3) on a minibatch shared by the generation. Algorithm 1 gives the operational order. Here $\mathcal{A}_T(g; A) = (c, \xi, U_T(g; A))$ denotes the adapted genome, with $\mathcal{A}_T(g; A) = g$ for an elite. Reproduction, adaptation, and evaluation with speciation form the conceptual composition

$$\mathcal{P}_{t+1} = \mathcal{S}_t\big(\{\mathcal{A}_T(g; A_t) : g \in \mathcal{R}(\mathcal{P}_t)\}\big). \tag{13}$$

Periodic grid updates then prepare the next generation.

Why adapt before selecting. For a fixed realization with Jacobian $\mathbf{J} = \partial \mathbf{b}/\partial \theta$, a gradient step moves behavior by

$$\Delta \mathbf{b} \approx -\eta\, \mathbf{J}\mathbf{J}^\top \nabla_{\mathbf{b}} \mathcal{L}. \tag{14}$$

The Jacobian depends on structure, realization *and* parameters. Equal predictions can therefore conceal different local learning directions. This motivates judging proposals after adaptation and retaining structural information alongside behavior during speciation. Innovation distance is only a structural proxy, not a Jacobian estimator. Equation (14) provides local intuition for gradient descent; the actual inner optimizer is Adam (Appendix E).

The cap covers search and both conversions together; it does not promise that every generation or candidate completes. Execution uses eager PyTorch autograd, with rows vectorized within an individual and individuals processed sequentially on one CPU thread per benchmark job. Appendix C explains the cost and deadline accounting.

A.4 Operators and the family each one searches

Structural invariant of crossover. C-KAN follows NEAT's fitter-parent inheritance rule (Stanley & Miikkulainen, 2002). Matching innovations inherit an entire spline payload from one parent, avoiding coefficient mixing across incompatible grids. Unmatched innovations come only from the fitter parent g_f (ties choose the first). Typed split innovations and a fitter-parent resolution of output-type conflicts preserve node types. Consequently, for genomes produced by these evolutionary operators,

$$c_{\text{child}} = c_f, \qquad \mathbf{b}_B(\text{child}) \in \mathcal{M}_{c_f}(B) \quad \text{when its predictions are finite.}$$

Whole-edge and type-consistent inheritance extend this NEAT-style invariant to heterogeneous spline realizations. Its significance is the *family in which recombination searches*: matched grids, parameters and enabled flags can differ from g_f, so the child can move far within $\mathcal{M}_{c_f}$ without leaving its historical template. It need not remain in the narrower $\mathcal{M}_{c_f,\xi_f}$, interpolate parental predictions, or sample the family uniformly. The historical graph is already acyclic, so no cycle repair

Table 5: Where each operator searches. Approximate transfers concern local fit errors, not uniform bounds on the complete network.

Operator	Family after the move	Effect at application
Elite copy	unchanged	identical state
Gradient adaptation	$\mathcal{M}_{c,\xi}$	loss-directed parameter update
SM-scaled perturbation	$\mathcal{M}_{c,\xi}$	sensitivity-scaled exploration
Edge reset	$\mathcal{M}_c$	new parameters, possibly new grid
Grid adaptation / extension	$\mathcal{M}_c$, new ξ	sampled curve transfer
Add node	new template c'	fitted identity insertion
Add connection to $\otimes$	new template c'	perturbed unit-function entry
Add connection to $\oplus$	new template c'	random additive contribution
Crossover	$\mathcal{M}_{c_f}$	parental payload recombination

is required. Appendix F gives the inheritance argument and mask rule; Table 5 places it alongside the other operators.

A.5 Safe Mutation, speciation and recovery: further discussion

On an edge with a quiet spline branch, coefficients can therefore look insensitive and hit the clamp $\sigma_{\min}$, and receive steps up to $1/\sigma_{\min}$ times the ordinary size. The change may initially have little effect, then become visible when $w_{s,e}$ grows. Additive initialization makes the spline branch quiet; its base branch need not be. The correction targets this latent amplification of shape changes. C-KAN divides the amplitude out *before* clamping, for spline coefficients only. For a perturbation $\zeta_j \sim \mathcal{U}(-s, s)$:

For $|w_{s,e}| \geq 10^{-12}$, the corrected value equals $\bar{\sigma}_{e,j}$, the output's sensitivity to the edge's *shape*; below that threshold, the denominator guard matters. It still carries the downstream factor $\partial f/\partial z_e$, so the step stays output-aware. The scales keep their raw sensitivities, since the gradient of $w_{s,e}$ already measures the amplitude's effect. A coefficient on an edge with $w_{s,e} = 0$ receives the ordinary step, never the clamped maximum.

The original SM-G-SUM reduction needs one vector–Jacobian product per output; SM-G-ABS avoids cancellation of opposite per-example derivatives but costs more in this implementation. We use SUM for its lower cost across a population of heterogeneous graphs. C-KAN's amplitude correction addresses the separate spline-specific coupling, and works with either reduction (Appendix G).

where E, D count excess and disjoint genes and N is NEAT's size normalizer. The term u is a normalized MSE between two individuals. It is scaled by the fixed training-target variance rather than by either genome's own variance, so it is symmetric and cannot divide by zero. Since $1-e^{-u} = u+O(u^2)$, nearby behaviors retain first-order sensitivity, while large discrepancies contribute at most c_B. Outlying predictions therefore cannot make the behavioral term arbitrarily large. The structural term records genealogical differences and the behavioral term differences in fitted functions; neither directly measures the Jacobian. The bounded dissimilarity is used only for species membership, alongside NEAT fitness sharing, stagnation and offspring allocation (Appendix G). It is not a novelty objective, and no triangle inequality is assumed.

Members are ranked by $\mathcal{L}_{\mathcal{D}_{\text{sel}}}$. With at least K eligible species, the champions of the K best species are taken. Otherwise each species contributes its champion, and the remaining slots are apportioned by inverse champion loss and topped up round-robin across species. Structural twins are kept: conversion failures are properties of fitted parameters, so twins can convert differently.

with $L_{\min}$ initialized to the candidate's holdout loss L_0. The reference is the best loss so far, not the latest, so every accepted state satisfies $\mathcal{L}_{\mathcal{D}_{\text{sel}}} \leq (1 + \tau_p)L_0 + \alpha_p$: allowed degradation cannot compound. This acceptance invariant concerns pruning; the subsequent active-fan-in snap and refit are separate steps. Preparation uses Adam, grid adaptation and L-BFGS, with rollback if refitting fails or worsens training loss.

the smallest expression tree within a fidelity band of the best candidate, or an explicit failure if $\mathcal{V} = \varnothing$. Every candidate's outcome is recorded, so the value of the population is measurable within

the same runs: how often s^* comes from a candidate other than the numerical champion, and how often the champion's attempted conversion failed (Appendix H). The mechanism is independent of the backend.

B Data flow, preprocessing, and objectives

The method receives a training set and a random seed. The benchmark adapter reserves a seed-determined 10% of these rows for the internal selection set $\mathcal{D}_{\text{sel}}$, using a random permutation and $n_{\text{sel}} = \max(1, \text{round}(0.1n))$; the rest form $\mathcal{D}_{\text{fit}}$. Official benchmark validation and test rows are never passed to the fitting routine, so internal candidate selection is distinct from the benchmark's later choice among algorithm seeds.

For each input coordinate, compute $a_j = \min_{\mathcal{D}_{\text{fit}}} x_j$, $b_j = \max_{\mathcal{D}_{\text{fit}}} x_j$, $\mu_{x,j} = (a_j + b_j)/2$ and $s_{x,j} = (b_j - a_j)/2$. Targets use $\mu_y = \text{mean}_{\mathcal{D}_{\text{fit}}} y$ and $s_{\text{out}} = \text{std}_{\mathcal{D}_{\text{fit}}} y$. A zero scale is replaced by one in both cases. Training uses

$$\widetilde{x}_j = (x_j - \mu_{x,j})/s_{x,j}, \qquad \widetilde{y} = (y - \mu_y)/s_{\text{out}}. \tag{15}$$

Inputs are thus min–max mapped onto the initial grid range $[-1, 1]$, not standardized. Holdout inputs can fall outside $[-1, 1]$ and are not clipped. The returned expression removes both transforms analytically:

$$s_{\text{original}}(x) = \mu_y + s_{\text{out}}\, \widetilde{s}\big((x - \mu_x) \oslash s_x\big). \tag{16}$$

All internal losses, sensitivities and behavioral distances refer to the transformed coordinates. The fitness scale s_y^2 is the population variance of the full $\mathcal{D}_{\text{fit}}$ target, computed once. It is therefore close to one under the adapter's standardization, but the library does not assume this. The floor 10^{-12} defines the constant-target case; scale invariance is not asserted when the floor is active.

Three distinct objectives. Adam updates and internal holdout decisions use MSE. Evolution uses MSE divided by the fixed training-target variance, plus structural parsimony. For Tracks B/C, the external benchmark selects seeds with the pointwise relative criterion of the official SRSD selector,

$$L^{\text{val}}_{\text{official}} = \frac{1}{n_{\text{val}}} \sum_{i=1}^{n_{\text{val}}} \left(\frac{s_{\text{original}}(x_i) - y_i}{y_i} \right)^2, \tag{17}$$

subject to the benchmark's recorded policy for zero targets and nonfinite values. This is neither the internal MSE nor $\sum_i (\widehat{y}_i - y_i)^2 / \sum_i y_i^2$. Each track's data and scoring rules belong to the experimental protocol; they must not be inferred from the method's use of the word *validation*.

Minibatches and buffers. One seeded minibatch A_t of $\min(n_{\text{batch}}, |\mathcal{D}_{\text{fit}}|)$ rows is drawn per generation and shared by every individual, for both the gradient steps and the fitness. It is not a disjoint refinement/evaluation split. Separately, one uniform sample of at most 256 rows of the *full* $\mathcal{D}_{\text{fit}}$ is drawn per generation and shared as every genome's input buffer. Grid adaptation, the behavior probe and Safe Mutation read only this buffer, so none of them sees batch-only rows. The fitness scale is likewise computed once from the full targets. Minibatching therefore shrinks only the fitness and gradient cost.

C Execution order and computational budget

Algorithm 1 is in Appendix A; Algorithm 2 specifies reproduction. Initialization is trained and scored before the first reproduction. Thereafter reproduction uses the previous generation's fitness and species. EMA drift precedes gradient adaptation, scoring follows adaptation, and grid relocation follows scoring and speciation. Elites skip mutation, drift, gradient adaptation and grid updates, but are evaluated again on each generation's shared minibatch. Thus elitism preserves a genome's state, not a monotone loss measured on changing samples. Generation-level internal holdout loss is a diagnostic in the main configuration, not an early-stopping selection rule.

What executes on the CPU. The implementation builds a sparse active phenotype in topological order and uses ordinary eager PyTorch operations and autograd. Samples are vectorized within each edge operation. Different genomes are processed sequentially; there is no batching across topologies, TorchScript compilation, or custom fused DAG kernel. Independent benchmark jobs can occupy different cores, but this does not accelerate a single job's measured wall time.

For P_t^{learn} non-elites at generation t, the gradient-update count is $T \sum_t P_t^{\text{learn}}$. At the maximum $P = 150$, $H = 100$ and $T = 5$, this is at most $75{,}000$ updates, before accounting for early termination and elites. Their cost depends on active edges, spline resolutions and batch size, rather than this count alone. Minibatching reduces fitness and gradient costs; grid and sensitivity operations use their own bounded buffers. The default $T = 5$ is a budget allocation, not a claim that each newly grown network is optimized in five steps. Learned parameters persist along surviving lineages; Adam's optimizer state is reset between generations.

Deadline semantics. The benchmark adapter assigns one 3600-second fitting deadline to preprocessing, evolution, candidate preparation, and both conversions. GSR runs first as the primary backend. The remaining time is not reset for vanilla, and paired outputs share the same evolution identifier. Library checks occur at generation or trial boundaries; the benchmark supervisor enforces the process budget if an operation does not return promptly. Cleanup or external evaluation can make controller elapsed time exceed fitting time, so those durations must be reported separately. A cap is a stopping rule, not evidence that all 100 generations and all K candidates finish.

A runtime report should therefore include completion and timeout fractions together with stage-time distributions. Existing records distinguish evolution, preparation and backend extraction; they do not isolate every Adam, speciation and grid-operation cost. Fine-grained profiling requires separate instrumentation and must not be inferred by dividing the generation time by the number of updates. No such profile is claimed here.

D KOLMOGOROV–ARNOLD NETWORKS AND SPLINE PRELIMINARIES

From the representation theorem to KANs. The Kolmogorov–Arnold representation theorem writes every continuous $f : [0, 1]^n \to \mathbb{R}$ as

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q\Big(\sum_{p=1}^{n} \varphi_{q,p}(x_p)\Big), \tag{18}$$

using only univariate continuous functions and addition. The inner and outer functions can be highly non-smooth. Liu et al. (2025) observe that functions arising in science usually have smooth, sparse compositional structure, and generalize Eq. (18) to layered networks of arbitrary width and depth whose edges carry learnable univariate functions. C-KAN drops the layered template as well: any DAG whose edges carry such functions is admissible, and its structure is searched.

Mean aggregation costs no expressiveness. In Eq. (18), substituting $\Phi_q = \Phi'_q/(2n + 1)$ and $\varphi_{q,p} = \varphi'_{q,p}/n$ turns both sums into means of rescaled functions. Mean aggregation therefore only changes the scale of the functions the network must learn. At fixed topology a positive multiplier is also absorbed by edge scales, which is why mean versus sum is not ablated. After a fan-in change, however, an immediate change of the multiplier rescales the existing computation, which is what the EMA of Eq. (5) avoids.

B-spline facts used in the paper. An edge's knot vector extends its grid $t_0 < \cdots < t_G$ by k equally spaced knots on each side, giving $G + k$ degree-k basis functions $B_{j,k}(\cdot; t)$ (de Boor, 1978). They are non-negative, each is supported on at most $k + 1$ consecutive spans, and all vanish outside $[t_{-k}, t_{G+k}]$. On the effective domain $D = [t_0, t_G]$ they form a partition of unity, $\sum_j B_{j,k}(x) = 1$, and reproduce every polynomial of degree at most k. Three consequences are used repeatedly:

- *Effective domain.* Outside D the partition-of-unity property is no longer guaranteed; beyond the extended support every basis function is zero, leaving w_b SiLU. This is the mechanism behind F1.

- *Bounded coefficient perturbations.* For any perturbation ξ of the coefficients and $x \in D$, $|\sum_j \xi_j B_{j,k}(x)| \le \max_j |\xi_j|$. This is used in Proposition 2.
- *Locality of sensitivity.* A coefficient affects the output only for inputs in its support. If no buffered input falls in that support, its Safe-Mutation sensitivity is exactly zero, and it receives the ordinary rather than an amplified step. Knowledge stored for input regions not visited this generation is therefore neither frozen nor blown up.

Discrete least-squares projection. Given a grid t and sample points U, write $\widehat{\Pi}_{t,U}[r]$ for the spline on t that minimizes $\sum_{u\in U}(s(u) - r(u))^2$. If U satisfies the Schoenberg–Whitney condition for t (points $u_{(1)} < \cdots < u_{(G+k)}$ with $B_{i,k}(u_{(i)}; t) \neq 0$), the design matrix has full column rank. The minimizer is then unique, and $\widehat{\Pi}_{t,U}$ *reproduces* splines on t: if r is already such a spline, it is returned exactly, as a function. Evenly spaced samples with several points per span satisfy the condition. The complement construction uses 100 points on grids of at most $G_{\max} = 20$ spans, and grid extension uses $\max(100, 4G')$. Write $\hat{\varepsilon}_t(r) = \sup_{x\in D} |r(x) - \widehat{\Pi}_{t,U}[r](x)|$ for the error of the operator actually run. For any spline s on t, reproduction gives $r - \widehat{\Pi}r = (I - \widehat{\Pi})(r - s)$. Hence $\hat{\varepsilon}_t(r) \le (1 + \|\widehat{\Pi}_{t,U}\|_\infty) \inf_s \sup_D |r - s|$, and for smooth r the infimum is $O(h^{k+1})$ in the mesh h (de Boor, 1978). We use this only as a rate heuristic: a resolution-independent bound on $\|\widehat{\Pi}_{t,U}\|_\infty$ is a property of the sampling geometry, which we do not prove. All bounds below are stated in terms of $\hat{\varepsilon}$ itself.

E Behavior families and the role of local adaptation

The template c fixes node identities, node types, and all historical connection genes, including disabled ones. The realization ξ fixes which connections are enabled, their knot grids and resolutions, and the current scale states. Eq. (11) defines $\mathcal{M}_{c,\xi}(B) \subseteq \mathcal{M}_c(B)$. We restrict both sets to finite predictions on B. They describe behavior on a finite reference sample, not equality of functions everywhere. Parameter symmetries and dead structure make the parametrization non-injective, and reachable sets can intersect; they need not be manifolds.

Which operators move where. A coefficient or scale perturbation and a gradient step with finite resulting predictions stay in $\mathcal{M}_{c,\xi}$. Grid adaptation, grid extension and edge resets change ξ (the knot vector, and with extension also G_e), so they stay in $\mathcal{M}_c$ but not necessarily in $\mathcal{M}_{c,\xi}$. Add-node and add-connection change c. Crossover produces a finite child in $\mathcal{M}_{c_f}$ by the structural invariant in Section 3.3. Cloning copies g exactly; the subsequent mutation determines where the clone moves. An elite copy is never mutated or adapted.

Post-adaptation quality is a property of the state, not of the behavior. Eq. (3) assigns each proposal the loss of the model with parameters $U_T(g)$, with the learned parameters written back. It does not assign one "eroded" loss to each behavior vector: two individuals with equal predictions on B can have different Jacobians and hence different futures under U_T. Optimizer state and sampling also matter. Adam is re-initialized for each non-elite individual in each generation, and gradients are clipped to norm 1.

For interpretation only, hold ξ fixed and let $\mathbf{b}(\theta)$ collect outputs on the sample used by the loss. One plain gradient step gives

$$\theta^+ = \theta - \eta \mathbf{J}^\top \nabla_{\mathbf{b}} \mathcal{L}, \qquad \mathbf{J} = \frac{\partial \mathbf{b}}{\partial \theta}, \tag{19}$$

$$\mathbf{b}(\theta^+) - \mathbf{b}(\theta) = -\eta \mathbf{J}\mathbf{J}^\top \nabla_{\mathbf{b}} \mathcal{L} + O(\eta^2), \tag{20}$$

under a locally bounded second derivative. The reachable tangent directions are $\mathrm{col}(\mathbf{J})$, and $\mathbf{J}\mathbf{J}^\top$ determines their conditioning. Both depend on the realization and the current parameters. Adam changes the update, so Eq. (20) does not describe its exact trajectory. No convexity, monotone improvement or recovery theorem follows.

Lamarckian versus Baldwinian inheritance. Writing θ^+ back (Lamarckian) makes the next generation start from the adapted point. Keeping θ_0 and using θ^+ only for fitness (Baldwinian) would

select for learnability without inheriting what was learned (Hinton & Nowlan, 1987; Whitley et al., 1994). C-KAN uses Lamarckian inheritance to retain fitted parameters across generations; it does not restart local fitting at every birth. The library also implements the Baldwinian variant, where grid adaptation reads the genome's own (unadapted) payload so that adapted values cannot leak into the genotype.

When does growth enlarge a family? Suppose every parameter of a parent realization admits a child reparameterization ι with $f_{c',\xi',\iota(\theta)}(x) = f_{c,\xi,\theta}(x)$ on B. Then $\mathcal{M}_{c,\xi}(B) \subseteq \mathcal{M}_{c',\xi'}(B)$ by substitution; ι need not be injective. A transfer that only matches one trained parent approximately establishes that parent and child are close, not that one family contains the other. C-KAN's growth operators pursue this local property. Fitted identities and non-nested grids do not justify an unconditional set-to-superset theorem, and we claim none.

F Graph representation, aggregation, and constructive operators

F.1 Genotype and active phenotype

Connection genes carry endpoints, an innovation number, an enabled flag, and the entire spline payload $(t_e, G_e, c_e, w_{b,e}, w_{s,e})$. Node types are fixed at birth, never converted. Node-split innovations are keyed by the split connection's innovation *and* the new node's type. Additive and multiplicative splits of the same edge are therefore structurally disjoint, so a hidden-node identifier can never carry conflicting types. Output identifiers are shared by all genomes. Each generation-zero genome draws its output types independently (multiplicative with probability $p_\times$), so parents can disagree on an output type.

The phenotype retains the nodes and edges on enabled paths from an input (or, after pruning, the bias source) to an output. Disabled or disconnected genes are inherited but contribute neither to the complexity penalty nor to the equation. The initial genome connects every input directly to every output and has no hidden nodes. Its edges use the add-connection initialization of their destination (below). There is no deletion mutation during evolution; irrelevant dependence is removed post hoc by pruning.

F.2 Aggregation normalization

Let $\psi_\mu(p) = p(p^2 + \epsilon_\times)^{(\mu-1)/2}$ with $0 < \mu \leq 1$ and $\epsilon_\times > 0$.

Proof of Proposition 1. Additive: with $\mu_v = 1/n_v$, $|h_v| \leq \frac{1}{n_v}\sum_e |z_e| \leq M$. Multiplicative: for $p \neq 0$ the exponent $(\mu-1)/2$ is non-positive, so $|\psi_\mu(p)| = |p|(p^2+\epsilon_\times)^{(\mu-1)/2} \leq |p|(p^2)^{(\mu-1)/2} = |p|^\mu$, and both sides vanish at $p = 0$. With $\mu_v = 1/n_v$ and $|p_v| \leq M^{n_v}$, this gives $|h_v| \leq M$. At $\mu = 1$ the exponent vanishes, $h_v = p_v$, and $|p_v| = M^{n_v}$ when every factor has magnitude M. □

This bound formalizes the scale control observed in preliminary tests: raw product nodes more often yielded exploding or nonfinite activations than geometric nodes. Normalizing the output of each node reduces amplification along later graph paths. It is a real-arithmetic bound, however, and the implementation evaluates p and p^2 before normalization. An extreme within-node intermediate can therefore still overflow or underflow, even if the final mathematical value is bounded. During an EMA transient the general bound is $|p|^\mu$ rather than M. The fitness and extraction checks screen nonfinite outputs, but do not prevent intermediate floating-point overflow.

For fixed μ,

$$\psi'_\mu(p) = (p^2 + \epsilon_\times)^{(\mu-3)/2}(\epsilon_\times + \mu p^2) > 0, \tag{21}$$

so ψ_μ is smooth, odd, strictly increasing and unbounded: a diffeomorphism of $\mathbb{R}$. An unrestricted downstream univariate function could compose with ψ_μ^{-1} and recover the raw product exactly, across sign changes. A finite spline or a finite symbolic library need not represent that inverse, and an output node has no downstream edge. Normalized multiplication is therefore an inductive bias with a representation trade-off, not a claim of exact equivalence to raw products in every finite

architecture. The two modes favor different representations: raw products express x_1x_2 directly, whereas the unsmoothed geometric form would express a signed root directly. Positive smoothing prevents identifying the implemented form with an exact square root. Raw-product mode (μ ignored) remains available and is ablated. For $\mu < 1$ and $\epsilon_\times > 0$, ψ_μ is not exactly $\mathrm{sign}(p)|p|^\mu$, which is why the exporter keeps the smoothed form. A flag optionally emits the clean power $(\prod)^\mu$ instead, which can domain-violate.

Scale dynamics. For a constant target count n, Eq. (5) has the solution $\mu^{(t+r)} = 1/n + \beta^r(\mu^{(t)} - 1/n)$. After an edge is added to an additive node with $n-1$ existing inputs, the per-generation rescaling factor $\mu^{(r+1)}/\mu^{(r)}$ lies in $[\frac{n-1}{n}, 1]$. Its deviation from one is at most $(1-\beta)\beta^r/(n-1)$, and the cumulative factor converges geometrically to $(n-1)/n$. At a geometric node, changing the exponent by Δ multiplies $|h_v|$ by $(p_v^2 + \epsilon_\times)^{\Delta/2}$, an activation-dependent factor. The EMA is applied once per generation to non-elites, whose lineages therefore drift even when an elite ancestor does not. During evolution the target counts *enabled* incoming genes, since no phenotype exists during crossover and building one per child would be costly. The extraction-time snap uses the active phenotype and is followed by a refit.

F.3 Initialization and approximate transfer under growth

Let $t^{(0)}$ be a uniform grid with $G_0 = 3$ intervals on $[-1, 1]$. The complement construction of Eq. (4) fits the spline branch by least squares to $r - \mathrm{SiLU}$ at 100 evenly spaced points, with $w_b = w_s = 1$.

- *Edge into* $\oplus$: $w_b \sim \mathcal{N}(0, \sigma_b^2)$, $w_s \sim \mathcal{N}(0, \sigma_s^2)$, $c_{e,j} \sim \mathcal{N}(0, \sigma_c^2)$, with $(\sigma_b, \sigma_s, \sigma_c) = (1, 0.01, 0.1)$. The edge starts close to a random-weight SiLU, with a small spline branch. The nonzero random factors permit coefficient and scale learning, although downstream gating can still make a particular gradient zero.
- *Edge into* $\otimes$: $\phi^{[1]}$ plus i.i.d. $\mathcal{N}(0, \sigma_c^2)$ jitter on the coefficients and both scales.
- *Split*: $\phi_{u\to w} = \phi^{[\mathrm{id}]}$, $\phi_{w\to v} = \phi_{u\to v}$ (copied with its grid), $\mu_w = 1$; the old gene is disabled, so n_v is unchanged and μ_v is left as it is.
- *Edge reset* (the non-perturbation branch of weight mutation): the destination's initialization above, which can also reset the grid.

The next proposition quantifies local changes at application, in exact arithmetic with the stated projection assumptions. It is stated for the operators as implemented, jitter included, and in terms of the projection error $\hat{\varepsilon}$ of Appendix D. Let $S_D = \sup_{x\in D} |\,\mathrm{SiLU}(x)|$.

Proposition 2 (Change at application)**.** *Assume every edge input stays in the effective domain of that edge, and hold the receiving scale μ_v fixed, as the operators do.*

(i) Add-connection into $\otimes$. *With jitters ξ_b, ξ_s on the scales and $\xi_1, \dots, \xi_{G+k}$ on the coefficients, the new edge satisfies, on D,*

$$|\phi_{e'}(x) - 1| \le \delta := \hat{\varepsilon}(1 - \mathrm{SiLU}) + |\xi_b| S_D + |\xi_s|\big(1 + S_D + \hat{\varepsilon}(1 - \mathrm{SiLU})\big) + (1 + |\xi_s|) \max_j |\xi_j|.$$

If $\delta < 1$, the node's product is multiplied by $1 + \delta'$ with $|\delta'| \le \delta$, and its activation by a factor between $(1+\delta')^{\mu_v}$ and $1 + \delta'$, with unchanged sign. For i.i.d. $\mathcal{N}(0, \sigma_c^2)$ jitter, $\max_j |\xi_j| = O\big(\sigma_c \sqrt{\log(G+k)}\big)$ with high probability.

(ii) Add-connection into $\oplus$. *The activation changes by $\mu_v z_{e'}$, where $z_{e'}$ has mean zero and variance $\sigma_b^2\,\mathrm{SiLU}(h_u)^2 + \sigma_s^2\sigma_c^2 \sum_j B_{j,k}(h_u)^2 \le \sigma_b^2\,\mathrm{SiLU}(h_u)^2 + \sigma_s^2\sigma_c^2$.*

(iii) Add-node. *The new node passes the incoming edge output unchanged; that edge only approximates the identity. The destination's fan-in is unchanged. The replacement path changes the old edge's output by at most $L_\phi\,\hat{\varepsilon}(\mathrm{id} - \mathrm{SiLU})$, where L_ϕ is a Lipschitz constant of the copied edge on the region traversed.*

(iv) Grid extension. *The edge changes on D by at most $|w_s|$ times the projection error of its old spline branch onto the finer grid. For a uniform grid doubled on the same core interval, the restricted spline spaces are nested; a full-rank exact least-squares solve then has zero transfer error on that interval. This is not a finite-precision guarantee.*

Proof. (i) The implemented edge is $(1+\xi_b)\,\mathrm{SiLU}+(1+\xi_s)\sum_j(c_j+\xi_j)B_{j,k}$, where c are the coefficients of $\widehat{\Pi}[1-\mathrm{SiLU}]$. Subtracting $1=\mathrm{SiLU}+(1-\mathrm{SiLU})$ gives $\phi_{e'}-1=\big(\widehat{\Pi}[1-\mathrm{SiLU}]-(1-\mathrm{SiLU})\big)+\xi_b\,\mathrm{SiLU}+\xi_s\widehat{\Pi}[1-\mathrm{SiLU}]+(1+\xi_s)\sum_j\xi_jB_{j,k}$. Bound the four terms by $\hat{\varepsilon}$, $|\xi_b|S_D$, $|\xi_s|(\sup_D|1-\mathrm{SiLU}|+\hat{\varepsilon})\le|\xi_s|(1+S_D+\hat{\varepsilon})$, and, by the partition of unity, $(1+|\xi_s|)\max_j|\xi_j|$. For the activation, $p'=p(1+\delta')$ and $h'/h=(1+\delta')\big((p'^2+\epsilon_\times)/(p^2+\epsilon_\times)\big)^{(\mu_v-1)/2}$. The ratio inside lies between 1 and $(1+\delta')^2$, and the exponent is non-positive, so h'/h lies between $(1+\delta')^{\mu_v}$ and $1+\delta'$. The Gaussian maximum is standard. (ii) The existing contributions are unchanged because μ_v is fixed. The weights w_b, w_s and c are independent and zero-mean, so the cross term vanishes, and $\sum_j B_j^2\le(\sum_j B_j)^2=1$ on D. (iii) The node w has one input and $\mu_w=1$, so $h_w=z_{u\to w}$ for either type ($p(p^2+\epsilon_\times)^0=p$). $|\phi^{[\mathrm{id}]}(x)-x|\le\hat{\varepsilon}(\mathrm{id}-\mathrm{SiLU})$ by Eq. (4), and the copied edge is Lipschitz on the region. (iv) The scales are unchanged, and the spline branch is replaced by its projection. If $G'=2G$ on a uniform grid over the same interval, every old knot is a new knot, so the old spline lies in the new spline space on D, and reproduction returns it exactly. □

Propagating these bounds to the network output requires downstream Lipschitz constants, which we do not assume uniformly. Inputs that leave an effective domain can make the actual change larger. The proposition makes the qualitative claim of Table 5 precise. Multiplicative entry and node insertion are preserving up to fitting and jitter terms. Grid extension is preserving up to a projection error, with the conditional nested-space case described above. Additive entry has a zero-mean local contribution; a nonlinear downstream graph need not preserve its output even in expectation.

Why not use all-zero additive entry? A zero contribution is the additive identity, so initializing an added edge that way would preserve the receiving activation at birth while its scale is held fixed. The evaluated configuration instead retains random-base exploration. This trades immediate preservation for a nonconstant new path; its superiority over zero entry is not established here. In particular, setting both the spline amplitude and all coefficients to zero yields zero gradients for both sides of that multiplicative parameterization: $\partial\phi/\partial c_j=w_sB_j=0$ and $\partial\phi/\partial w_s=\sum_j c_jB_j=0$. The base scale can still learn, and mutation could later reactivate the spline, but all-zero initialization is not an interchangeable implementation detail. A preservation-oriented alternative would need an initialization that retains useful learning directions.

F.4 Resolution mutation and domain adaptation

Grid extension passes a genome-level gate ($p=0.05$), then an edge-level gate (0.1) for each enabled edge. It samples G'_e uniformly from $\{G_e+1,\dots,\min(2G_e,G_{\max})\}$, builds a uniform grid on the edge's existing active interval, and transfers the coefficients by

$$c'_e\in\operatorname*{argmin}_a\sum_{u\in U_e}\Big[\sum_j a_jB_{j,k}(u;t'_e)-\sum_j c_{e,j}B_{j,k}(u;t_e)\Big]^2 \tag{22}$$

over $|U_e|=\max(100,4G'_e)$ evenly spaced points. The scales are unchanged. The new grid need not contain the old knots, so this is not exact knot insertion (de Boor, 1978), and preservation is up to the projection error.

Grid *adaptation* has a different purpose: it moves an existing grid to where the edge's inputs lie. Every $T_{\mathrm{grid}}=3$ generations, for each non-elite genome and each active spline edge, the n_B buffered inputs $u_{(1)}\le\cdots\le u_{(n_B)}$ of the edge define the knots

$$t_i=(1-\gamma)\,q_i+\gamma\Big(u_{(1)}+\tfrac{i}{G_e}\big(u_{(n_B)}-u_{(1)}\big)\Big),\qquad i=0,\dots,G_e, \tag{23}$$

where $q_i=u_{(1+\lfloor in_B/G_e\rfloor)}$ for $i<G_e$ and $q_{G_e}=u_{(n_B)}$. This is a mostly quantile-spaced grid with a small uniform component ($\gamma=0.02$). The current curve, evaluated at the buffered inputs, is then refit on the new grid. Edges with fewer than 10 buffered inputs, or with input variance below 10^{-6}, keep their grid.

F.5 Crossover within a parental template

Argument for the crossover invariant. Genes are aligned by innovation number. A gene of g_f is either matching (present in both parents) or disjoint/excess of g_f; both kinds are inherited. A

disjoint or excess gene of the other parent is dropped. The child's set of connection identities is therefore exactly g_f's; their payloads can differ. Hidden nodes are included exactly when they are endpoints of inherited genes. Every hidden node of g_f is the endpoint of the two genes created by the split that introduced it, and genes are never deleted during evolution, so the child's node set is g_f's. Hidden types agree between parents by the split-keying rule; for a type-conflicted output, the type and all incoming genes are taken from g_f. Hence $c_{\text{child}} = c_f$. Add-connection rejects a cycle in the *full* historical graph, disabled genes included, so c_f is acyclic, and so is every enabled subset of it. If its predictions on B are finite, the child lies in $\mathcal{M}_{c_f}(B)$ by definition. This argument concerns evolutionary genomes whose hidden nodes retain their birth connections, not arbitrary hand-edited graphs with orphaned node records.

A matched connection takes one parent's entire payload, including its grid and resolution. If both copies are enabled, the child copy is enabled; otherwise it is disabled with probability $p_d = 0.75$. The enabled graph can therefore differ from g_f's in both directions. After assembly, the child recomputes its fan-ins and snaps every μ_v to $1/n_v$. A recombinant lies in $\mathcal{M}_{c_f}$ until structural mutation acts. It need not lie in $\mathcal{M}_{c_f;\xi_f}$, be close to either parent, interpolate their predictions, or sample the family uniformly. This is the precise scope of the statement that crossover places the child "somewhere in the fitter parent's family".

G Functional variation, speciation, and population selection

Amplitude-corrected Safe Mutation. Sensitivities are evaluated on a temporary phenotype of the child after crossover or cloning and any grid extension, but before coefficient perturbation, so the divisors match the child's actual parameter shapes and grids. Inputs come from the parents' pooled buffers, deduplicated by identity so that rows shared by both parents are not double-weighted, and subsampled to at most 256. For multiple outputs,

$$\sigma_j = \Big[\sum_o \Big(\sum_{x \in B_{\text{SM}}} \frac{\partial f_{g,o}(x)}{\partial \theta_j} \Big)^2 \Big]^{1/2}. \tag{24}$$

Parameters are matched to edges by innovation number, never by position. SM-G-SUM and SM-G-ABS are both output-gradient reductions from Lehman et al. (2018); the former sums signed sensitivities, while the latter averages their per-example magnitudes. By Eq. (6), $|w_{s,e}|$ is constant over the batch, so it factors exactly out of either reduction. The correction of Eq. (7) therefore applies unchanged to both, and neither reduction nor $\sigma_{\min} = 0.01$ is modified. The guard 10^{-12} only prevents division by zero: an exactly zero amplitude yields exactly zero coefficient sensitivities, which the zero map sends to the ordinary step. Parameters outside the active subgraph, and children whose parents have no buffered inputs, receive ordinary perturbations. For scalar regression, SUM differentiates $\sum_{x \in B_{\text{SM}}} f_g(x)$ in one vector–Jacobian product. The ABS implementation instead loops over examples and differentiates their outputs before taking absolute values. Thus it requires up to $|B_{\text{SM}}|$ such products per output; an ordinary training gradient does not supply these per-example output derivatives.

The original SM-G-SUM reduction can cancel: if two rows have derivatives a and $-a$, their summed sensitivity is zero although both predictions respond to that coefficient. An exactly zero divisor takes the ordinary step; a small nonzero divisor reaches the floor and can amplify it. SM-G-ABS avoids this cancellation, with a different computational cost and sensitivity scale. We use SUM for efficiency; C-KAN's spline-amplitude normalization is compatible with either borrowed reduction and solves a separate problem. Symbolic edges, which exist only after extraction starts, are never mutated.

Two residual gating routes are not corrected, by design, and are reported for completeness. First, an edge into a *raw* product node is scaled by the product of its siblings, so a near-zero sibling makes the whole edge look insensitive. Geometric normalization changes this gating but does not eliminate it, especially near zero with positive smoothing. Second, the scale parameters keep raw sensitivities and can be gated by a flat downstream edge. Both are instances of the same pattern across edges rather than within one.

Mutation pipeline. Each non-elite child passes, in order, through: grid extension (above); weight mutation (genome gate 0.8; then, per edge, a perturbation with probability 0.9 or else a reset),

Algorithm 2 REPRODUCE: NEAT population management with C-KAN operators

Require: population $\mathcal{P}$ with fitnesses and species
compute shared qualities $\widehat{q}$; drop stagnant species; refocus if globally stagnant
allocate offspring: reserve 2 per species, remainder $\propto \sum \widehat{q}$
for each species s with quota n_s **do**
    truncate to the top 20% as the parent pool
    if $|s| > 5$ **then**
        copy the champion unchanged (elite)
    end if
    while s has unfilled quota **do**
        if $\mathrm{rand} < 0.75$ **then**
            child $\leftarrow$ crossover of two pool parents (§3.3); snap μ
        else
            child $\leftarrow$ clone of one pool parent
        end if
        grid extension; SM-scaled weight mutation (Eq. (7)); add-connection; add-node
    end while
end for
return next population

where coefficients receive $\mathcal{U}(-0.5, 0.5)$ steps and scales $\mathcal{U}(-0.5, 0.5)$ steps before SM scaling; add-connection (0.08); and add-node (0.03, the new node multiplicative with probability $p_\times$).

Behavioral probe and compatibility. The behavior probe B_t contains up to 64 inputs drawn uniformly from the generation's shared training buffer. It is shared by every individual and by every species representative (a representative carried over from the previous generation is re-evaluated on B_t). Floating-point evaluation of Eq. (8) uses $-\mathrm{expm1}(-u)$ to avoid cancellation. For multiple outputs, u averages over all output coordinates. The term is symmetric, satisfies $1 - e^{-u} = u + O(u^2)$, and is bounded by one. Equality on a finite probe does not imply equality elsewhere, and the triangle inequality is not guaranteed. Excess and disjoint counts include historical (disabled) genes. The normalizer N is the larger connection count, except that genomes below 20 connections use $N = 1$. KAN genomes use no coefficient-distance term. An individual with invalid fitness, or a non-finite behavioral distance, is compared by the structural terms only. Species representatives are drawn from valid members. Invalid individuals never found a species: they join the first compatible species, or else the structurally nearest one.

Fitness sharing and offspring allocation. Fitness in Eq. (3) is generally negative, so valid fitnesses are min–max scaled to a non-negative quality before explicit sharing,

$$q_i = \frac{F_i - F_{\min}}{F_{\max} - F_{\min}}, \qquad \widehat{q}_i = q_i / n^{\mathrm{valid}}_{s(i)}, \tag{25}$$

with $q_i = 1$ when all valid fitnesses are equal. Invalid genomes receive zero. Each non-stagnant species first reserves 2 offspring. The remainder is distributed in proportion to the species' summed $\widehat{q}$ with largest-remainder rounding; if the reserve cannot be met, the population is split evenly. A species that has not improved for 15 generations stops reproducing. If the whole population has not improved for 20 generations, reproduction is refocused on the two best species. Within a species, members are truncated to the top 20%. A species with more than 5 members keeps its champion unchanged (elitism). Other children are produced by crossover with probability 0.75 (the second parent is drawn from another species with probability 0.001), or else by cloning, and are then mutated. Invalid genomes are never parents, champions or candidates. These mechanisms are NEAT's population management, not C-KAN novelty claims.

H POPULATION-BACKED SYMBOLIC RECOVERY

H.1 CANDIDATE ALLOCATION AND PREPARATION

Candidates are drawn from the final population, excluding invalid genomes and non-finite holdout losses. Each species' members are ordered by MSE on $\mathcal{D}_{\text{sel}}$. With at least K eligible species, the champions of the K species with lowest champion loss are chosen. With fewer species, one slot is reserved per species and the remaining slots are distributed with weights

$$w_s = \frac{1}{L_s^{\text{champion}} + 10^{-12}}, \tag{26}$$

using largest-remainder rounding. This has the reserve-first shape of reproductive allocation, but uses inverse holdout loss rather than reproductive fitness. If a quota exceeds a species' size, the remaining slots are filled round-robin across unexhausted species, so an exhausted lineage hands its slot to another lineage rather than to a deeper member of the best one. The selected candidates are finally ordered by holdout loss; the recorded rank is within this stratified list, not necessarily the global population rank.

Attribution order. Pruning order follows KAN attribution adapted to a sparse DAG. With A_e the standard deviation of edge e's output over $\mathcal{D}_{\text{fit}}$ and A_v the standard deviation of node v's *pre-aggregation* value (the sum, or the raw product), scores are seeded as $S_o = 1$ at each output and propagated backward in reverse topological order,

$$S_{e:u\to v} = \frac{A_e}{A_v + 10^{-4}}\, S_v, \qquad S_u = \sum_{e\in \text{Out}(u)} S_e, \tag{27}$$

averaged over outputs. Using the pre-aggregation value rather than the activation avoids biasing the ratio against high-fan-in nodes, since the multiplier scales the activation by $1/n_v$. At multiplicative nodes the additive derivation does not strictly hold, so there it serves as a heuristic ordering. The score only orders trials, so a misordering can only lead the greedy search down a worse path; acceptance still has to satisfy Eq. (9).

Pruning trials. The lowest-scoring untried spline edge is replaced by a new edge from a unit-valued bias node. This bias node is created only after evolution and is invisible to mutation, crossover and distance. Each replacement has its own constant edge, even when it uses the same unit bias source. Merging these graph edges before the active-fan-in snap would change the count and hence the scale or root exponent. Constants may instead be algebraically combined after equation composition, retaining the fixed scale and smoothing term; prediction-checked simplification is the appropriate place for that operation. Upstream structure that loses all output paths becomes inactive. A trial applies 30 Adam steps on $\mathcal{D}_{\text{fit}}$ (learning rate 10^{-2}) to the exact trial graph and evaluates it on $\mathcal{D}_{\text{sel}}$. Acceptance copies both the trial structure and its fitted parameters; rejection restores the committed state and permanently excludes the edge. Attribution is recomputed from the committed model. Trials stop when no eligible edge remains, a round limit is reached, or the deadline passes.

For Eq. (9), $L_{\min}^{(t)} \le L_0$ and never increases, so every accepted trial satisfies $L_t \le (1+\tau_p)L_0 + \alpha_p$ with $(\tau_p, \alpha_p) = (0.02, 10^{-12})$. This invariant is stronger than applying the tolerance to the previous, possibly worse, trial. It concerns pruning acceptance only; the subsequent snap and refit are a separate stage. Mean replacement preserves a representative local contribution and the local factor count at initialization. It does not preserve the full downstream prediction under nonlinear composition. The refit may move the constant while keeping it independent of the removed input.

After pruning, scale states are snapped over the active edges. Numerical preparation then applies 100 Adam steps, grid adaptation, and 20 L-BFGS steps on $\mathcal{D}_{\text{fit}}$. A refit that is non-finite, or that worsens the training loss, is rolled back, restoring parameters and grids together. A worse holdout loss alone does not trigger rollback.

H.2 Backends and ordinary mathematical meaning

For each prepared candidate, vanilla and GSR conversion start from separate deep copies, so both start from the same pruned network. A primitive is represented as

$$\phi_e^{\mathrm{sym}}(z) = a_e\,\psi_e(b_e z + c_e) + d_e, \qquad \psi_e \in \mathcal{H} = \{x, x^2, x^3, 1/x, \sqrt{x}, \sin, \cos, \exp, \log, \tanh\}, \tag{28}$$

with affine parameters fitted to the edge. Constants are not in the library; they arise from affines and from bias edges. Converted edges keep their spline payload, so a rejected GSR trial reverts exactly.

Vanilla conversion fits each edge in isolation. A linear-first rule (Knottenbelt et al., 2024) accepts a sufficiently accurate linear fit ($R^2 \geq 0.99$) before the full library is tried. The reason is that, for any smooth ψ,

$$a\,\psi(bz + c) + d = \big(a\psi(c) + d\big) + a\,b\,\psi'(c)\,z + O(b^2 z^2), \tag{29}$$

so on a narrow domain every primitive with $\psi'(c) \neq 0$ matches a linear edge to arbitrary accuracy, given sufficiently extreme affine parameters. Selecting by R^2 among such near-ties is then decided by noise, and yields obfuscated terms whose apparent linearity rests on cancelling large constants. Testing the line first breaks the tie in favour of the simplest form. Remaining choices trade local fit against operator complexity, ranked by KAN's primitive ordering, with a $+1$ penalty for domain-restricted operators. GSR is the method of Sovrano et al. (2026). Our adaptation warm-starts each trial primitive from its edge fit, so GSR considers the same candidates from the same start as vanilla. It fine-tunes the composed model for 100 steps on $\mathcal{D}_{\mathrm{fit}}$, scores it on $\mathcal{D}_{\mathrm{sel}}$, and commits the complete winning state. All trials for an edge begin from the same snapshot. Grids are adapted after a commit, never between competing trials. A candidate enters a 5% relative fidelity band around the edge's best trial loss, and the band is resolved by minimal operator complexity. This lexicographic rule is our choice: it applies simplicity to end-to-end loss, not to local R^2. The linear-first rule is not applied inside GSR, so the two backends remain separable for ablation. GSR is not a C-KAN contribution, and it is not guaranteed to outperform vanilla.

After conversion, joint affine refitting applies 300 Adam steps and then 50 L-BFGS steps. The intervening grid-update hook is a no-op once every edge is symbolic, and conversion is forced to completion because a spline/symbolic hybrid has no closed form. Guarded operators may stabilize optimization, but the exported formula uses the ordinary meaning of logarithm, square root and reciprocal. Validity is decided on that formula in float64, not on a guarded or float32 surrogate. The gates are, in order:

1. conversion must be complete;
2. parameters must be finite;
3. the exported equation must be finite on $\mathcal{D}_{\mathrm{fit}}$ and on $\mathcal{D}_{\mathrm{sel}}$;
4. the equation must agree with the float64 phenotype on rows where that reference is finite ($\mathrm{rtol} = 10^{-5}$, $\mathrm{atol} = 10^{-6}$); nonfinite reference rows are excluded and cannot by themselves reject a finite equation;
5. losses must be finite.

Per-row NaN and infinity rates are recorded for evaluable exported equations. The CSV label *domain-violation rate* counts all nonfinite predictions, including numerical overflow as well as genuinely undefined real expressions. Those counts alone do not identify the mathematical cause. Finite internal predictions do not guarantee validity on external validation or test inputs.

H.3 Selection, deadlines, and reporting

For each backend independently, Eq. (10) forms a relative band $\tau_e = 0.05$ plus an absolute tolerance $\alpha_e = 10^{-12}$ around the lowest valid symbolic holdout loss, then minimizes expression-tree size (the count of nodes in the expression tree), with loss breaking ties. Every completed candidate within the budget is considered; the procedure does not stop at the first valid equation. Size is measured after bounded, prediction-checked simplification. The simplifier runs in a subprocess that is killed on timeout, and when it cannot return a prediction-equivalent result the raw expression size is used. This library simplification is distinct from the benchmark's later canonicalization and NED evaluation, which have their own timeout and failure outcomes.

Measuring the value of the population. Candidate records identify whether the returned equation came from the numerical champion and whether the champion's attempted conversion passed the internal gates. Let $R_m \in \{1, \dots, K\}$ be the returned candidate's rank within the stratified list, and set $R_m = \infty$ when no equation is returned. For a fixed backend and M eligible attempts, report

$$\widehat{F}_R(k) = \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}[R_m \le k], \qquad \widehat{F}_{R|\mathrm{return}}(k) = \frac{\sum_m \mathbf{1}[R_m \le k]}{\sum_m \mathbf{1}[R_m < \infty]}. \tag{30}$$

The conditional quantity is undefined if no run returns an equation. The unconditional CDF retains failures; the conditional CDF describes ranks among returned equations. Rank is within the stratified shortlist, not necessarily the whole-population ordering. Internal conversion success is distinct from external benchmark validity or NED recovery.

A rank above one can mean that the champion failed, or that a later candidate was smaller within the fidelity band. Report these cases separately. If the deadline prevented a champion conversion from being attempted or recorded, its validity is missing, not a demonstrated domain failure. These diagnostics do not reproduce a new run with smaller K: changing K can alter species quotas, preparation order and remaining conversion time. They guide candidate-count ablations but do not replace them.

The main configuration uses one wall-clock fitting budget of 3600 seconds, shared by preprocessing, evolution, candidate preparation and both symbolic modes. GSR, the primary mode, runs first, and each mode's result is handed to the caller as soon as that mode completes. Pruning and GSR check the deadline before every trial and record a timeout locally rather than raising. The two modes share one population and are paired outcomes, not independent runs. A missing vanilla result can therefore reflect the shared budget rather than vanilla's own cost. Post-fit benchmark evaluation has a separate allowance.

I Relation to prior algorithms

Table 6 places C-KAN among the algorithms it borrows from or resembles. The discussion below states what is shared and what differs.

Table 6: C-KAN compared with related search paradigms. "Growth": the search includes structure-adding moves. "Pop.": population-based. "GD": gradient training inside the search loop. "Readout": how a model becomes the output.

Method	Searched object	Growth	Pop.	GD	Readout
KAN / KAN 2.0	coefficients of a fixed template	no	no	yes	prune, symbolify one model
NEAT	topology + scalar weights	yes	yes	no	best network
Prior NEAT+KAN (RL)	topology + splines	yes	yes	no	best policy
ESGD, PBT	weights / hyperparameters	no	yes	yes	best network
LEMONADE	CNN architecture	yes (morphisms)	yes	yes	Pareto front
Net2Net, morphisms, GradMax	one network's width/depth	yes	no	yes	grown network
Cascade-correlation	hidden units, frozen	yes	no	yes	grown network
DARTS	mixing weights of a supernet	no	no	yes	discretized cell
Memetic GP-SR	expression trees + constants	yes (syntax)	yes	local	best expression
C-KAN	typed sparse DAG + splines + resolution	yes	yes	yes	species-stratified symbolic recovery

KAN and NEAT. KAN supplies spline parameterization and curve-transfer machinery; KAN 2.0 provides explicit multiplication in a prescribed layered layout (Liu et al., 2025; 2024). C-KAN searches sparse typed DAGs and edge resolutions, normalizes changing fan-in, and uses population members for recovery. NEAT supplies historical markings, fitter-parent inheritance, speciation and complexification (Stanley & Miikkulainen, 2002). These are foundations, not new contributions.

The KAN-specific changes concern whole spline payloads, typed split identities, activation-range shifts, amplitude-sensitive mutation and equation recovery. Prior NEAT+KAN work already evolves KAN structure and functions for control (Nguyen et al., 2025); its conference abstract establishes that precedent but not an exhaustive inventory of its operators.

Hybrid evolution–gradient methods. ESGD interleaves gradient optimization and population selection (Cui et al., 2018); PBT transfers trained weights while adapting training hyperparameters (Jaderberg et al., 2017). LEMONADE combines multi-objective architecture evolution with approximate network morphisms and Lamarckian inheritance (Elsken et al., 2019). C-KAN shares this division of labor and weight reuse. It applies them to function-valued sparse graphs, prices active edges and spline intervals through scalar fitness, and retains species for symbolic recovery. The truncated objective describes that allocation of computation, not a new solution concept for bilevel optimization.

Function-preserving growth. Net2Net and network morphism transfer a trained function to a larger architecture (Chen et al., 2016; Wei et al., 2016); GradMax initializes growth using gradient information (Evci et al., 2022). C-KAN's fitted identity splits and grid projections have related transfer goals, but are approximate, and additive connection birth is exploratory. C-KAN does not claim an exact morphism for every structural move. Equation (4) specifies what replaces a scalar identity for its spline edge.

Differentiable architecture search. DARTS optimizes a continuous mixture over candidate operations in a supernet and later discretizes it (Liu et al., 2019a). C-KAN trains each proposed discrete graph directly. It avoids that architecture-relaxation mismatch but still faces a separate numerical-to-symbolic conversion gap. Population-backed recovery addresses this latter gap; it does not remove it.

Genetic-programming symbolic regression. Operon and PySR evolve expression structures with local constant optimization (Burlacu et al., 2020; Cranmer, 2023). C-KAN shares this evolutionary outer loop but delays commitment to primitives until after spline-graph search. Behavioral comparisons also have precedents in semantic GP (Moraglio et al., 2012). C-KAN uses its bounded dissimilarity for compatibility, not as a novelty objective or quality-diversity archive (Lehman & Stanley, 2011; Mouret & Clune, 2015). These connections locate the contribution in the representation and its coordinated search-and-recovery mechanisms.

J REFERENCE CONFIGURATION AND REPRODUCIBILITY DETAILS

Table 7 describes the current main `ckan_k5` configuration, not older pilot defaults. The library's standalone example and its adapter defaults can differ from track overrides. Pilot ablations with other generation counts, candidate counts or budgets must be identified separately in the experimental section; this table does not assert that historical runs used the current settings.

Table 7: Current main C-KAN configuration. Internal losses use transformed targets.

Component	Setting
Population / maximum generations	150 / 100
Shared generation minibatch	$\min(1024, \lvert\mathcal{D}_{\text{fit}}\rvert)$
Non-elite local learning	5 Adam steps, learning rate 10^{-2}
Inheritance / gradient clipping	Lamarckian / norm 1
Spline degree / initial and maximum intervals	3 / 3 and 20
Initial grid range / grid blend γ	$[-1, 1]$ / 0.02
Grid adaptation frequency / EMA β	every 3 generations / 0.5
Additive-edge initialization $(\sigma_b, \sigma_s, \sigma_c)$	$(1,\ 0.01,\ 0.1)$
Grid-extension gates (genome / edge)	0.05 / 0.1
Add-connection / add-node probabilities	0.08 / 0.03
Multiplicative-node birth probability	0.2; geometric mode, $\epsilon_\times = 10^{-8}$
Weight-mutation / conditional perturbation	0.8 / 0.9
Coefficient and scale perturbation half-widths	0.5 / 0.5 before SM scaling
Safe mutation / sensitivity floor $\sigma_{\min}$	SUM with amplitude correction / 0.01
Compatibility (c_E, c_D, c_B) / threshold	$(1, 1, 1)$ / 3
Experience buffer / behavior probe	256 / 64 training inputs
Crossover / interspecies mating / disabled-inherit	0.75 / 0.001 / 0.75
Survival fraction / elitism species size / species floor	0.2 / > 5 / 2
Species / global stagnation thresholds	15 / 20 generations
Parsimony $(\lambda_G, \lambda_E, \lambda_V)$	$(10^{-4}, 10^{-3}, 0)$
Candidate count K	5
Pruning relative / absolute tolerance	0.02 / 10^{-12}
Pruning trial steps / learning rate	30 Adam / 10^{-2}
Preparation refit (Adam / L-BFGS steps)	100 / 20
GSR trial fine-tuning / learning rate	100 steps / 10^{-2}
GSR and final fidelity tolerances	relative 0.05, absolute 10^{-12}
Post-conversion refit (Adam / L-BFGS steps)	300 / 50
Symbolic simplification allowance	up to 120 s, within the remaining fit budget
Shared fitting budget / execution	3600 s / CPU, one thread per job

Implementation accounting. Generation logs record structural size, species counts (all and valid) and fitness. Candidate records keep numerical rank, conversion outcome, equation, internal losses, domain-violation and non-finite rates, and preparation and refit timings. Summary records identify the selected candidate for each backend. A shared evolution identifier links paired modes, and configuration hashes distinguish method versions for downstream analysis; the analysis must still select compatible records explicitly. Benchmark validity, canonicalized size, NED, recovery and external prediction scores are computed outside the library from the exported original-scale equation, separately from the internal candidate diagnostics.

Scope of the claims. Constructive growth removes the need to choose one dense hidden-layer template, but it still has finite population, capacity, generation and runtime limits. Parsimony encourages compact active graphs without guaranteeing the shortest symbolic expression. Behavioral diversity and multiple candidates create opportunities to recover from conversion failures. They guarantee neither recovery nor test-domain validity, nor superiority over other symbolic regressors. Propositions 1 and 2, the crossover invariant, and the pruning acceptance rule are properties of specific steps, not convergence results for the evolutionary procedure.

K Extended related work

Symbolic regression. Genetic programming searches expression structure directly (Koza, 1992; Schmidt & Lipson, 2009); Operon and PySR pair evolutionary search with numerical constant optimization (Burlacu et al., 2020; Cranmer, 2023). Physics-informed decomposition exploits properties such as separability and symmetry (Udrescu & Tegmark, 2020). Learned policies and transformers propose expressions, sometimes combined with planning (Petersen et al., 2021; Kamienny et al., 2022; Shojaee et al., 2023); other methods distill trained networks into symbolic components (Cranmer et al., 2020). C-KAN belongs to this last broad strategy but evolves the continuous model's graph before assigning symbolic primitives. SRBench and SRSD supply complementary tests of predictive performance and equation recovery (La Cava et al., 2021; Matsubara et al., 2024).

KAN-based symbolic modeling. KAN contributes the spline parameterization, grid adaptation and numerical-to-symbolic workflow (Liu et al., 2025); KAN 2.0 adds explicit multiplication in a user-specified architecture (Liu et al., 2024). Mean aggregation has been studied as a way to moderate activation scale (Altarabichi, 2024). C-KAN adapts these ingredients to changing fan-in and evolved multiplication locations. For recovery, it uses linear-first edge fitting inspired by CoxKAN (Knottenbelt et al., 2024) and greedy in-context symbolic regression (GSR), which assesses substitutions through the composed model (Sovrano et al., 2026). These backends are credited components, with our fitting and selection adaptations specified in Appendix H. KAN-SR uses a trained KAN to guide decomposition through separability and translational symmetry (Bühler & Guillén-Gosálbez, 2025). C-KAN instead concentrates on constructing the intermediate graph and exploiting multiple fitted candidates.

Neuroevolution and growth. NEAT established historical alignment, speciation and complexification from minimal networks (Stanley & Miikkulainen, 2002). Prior work already combines NEAT with KAN structure and functional mappings for control tasks (Nguyen et al., 2025). Building on this precedent, C-KAN targets symbolic regression through specific variation, normalization, diversity and recovery mechanisms. Constructive architectures also have a long non-evolutionary history: cascade-correlation adds and freezes units (Fahlman & Lebiere, 1990); Net2Net and network morphism transfer trained functions into larger models (Chen et al., 2016; Wei et al., 2016); splitting methods and GradMax use gradient information to guide growth (Liu et al., 2019b; Evci et al., 2022). C-KAN retains their motivation to reuse learned behavior, but its spline transfers are approximate and not every structural mutation is function-preserving.

Hybrid search and functional diversity. ESGD combines gradient optimization with population selection (Cui et al., 2018), and PBT transfers trained weights while adapting training hyperparameters (Jaderberg et al., 2017). LEMONADE is a particularly close architectural analogue: approximate network morphisms warm-start offspring under multi-objective evolutionary search (Elsken et al., 2019). C-KAN also inherits trained parameters, but searches sparse function-valued graphs and recovers equations from a stratified population. Its structural decisions remain discrete, unlike DARTS' continuous relaxation (Liu et al., 2019a).

Safe Mutation already uses output gradients to scale perturbations (Lehman et al., 2018); C-KAN corrects the spline-specific amplitude factor in those sensitivities. Behavioral comparisons have precedents in semantic genetic programming, novelty search and quality-diversity methods (Moraglio et al., 2012; Lehman & Stanley, 2011; Mouret & Clune, 2015). Here behavior is a bounded compatibility term, not an additional optimization objective or an archive. Appendix I summarizes the searched objects and the role of local learning across these related paradigms.

L Benchmark protocol

This appendix gives the complete protocol summarized in Section 4.1.

L.1 Benchmark design

We evaluate three different questions rather than treating numerical fit and equation recovery as interchangeable. Track A asks how well a method predicts held-out SRBench data (Aldeia et al.,

2025). Track B asks whether it recovers the skeleton of a known equation from clean SRSD-Feynman observations (Matsubara et al., 2024). Track C repeats Track B on the *same equations* after adding measurement noise and irrelevant input columns. This last track is a controlled robustness test on synthetic data, not an experiment on observational real-world data. Table 8 gives the frozen main-paper profiles. The collectors also support larger rosters, but the short profiles and their denominators define the analyses described here.

Table 8: Main-paper benchmark profiles. BB denotes black-box and FP first-principles data. An algorithm seed is a repeated search attempt, not an independent dataset.

Track	Datasets	Seeds	Primary question
A	6 BB + 6 FP	10	Held-out prediction
B	10 Easy + 10 Medium + 10 Hard	5	Clean skeleton recovery
C	Same 30 equations as B	5	Recovery under joint corruption

Track A: predictive performance. We use six BB and six FP SRBench datasets, with the first ten frozen SRBench seeds (Appendix M.1). Each seed determines a 75/25 outer train/test split and any subsampling of the training partition to at most 40,000 rows. Feature and target standardizers are fitted on outer-training data; reported test errors use predictions transformed back to the original target scale. No noise or dummy columns are added. Where the archived wrapper has an outer hyperparameter grid, it uses three training-only folds; methods without such a grid retain their frozen native selection procedures. The held-out test set is never used to tune a method or choose a returned equation. The generic tuning allowance is at most six hours, followed by a one-hour final fit; methods without outer tuning do not use that allowance.

Track B: clean equation recovery. We use the official SRSD-Feynman train, validation and test files without resplitting: each equation has 8,000, 1,000 and 1,000 rows, respectively, plus a ground-truth expression. The short panel freezes ten Easy, ten Medium and ten Hard equations and the first five algorithm seeds (Appendix M.1). It is a deliberately curated subset of the 120-equation benchmark, not a random estimate of performance over all 120 equations. Methods fit on training data only. After each seed returns an equation, we evaluate its original-coordinate expression on the official validation set and select one seed per method and equation by pointwise mean squared relative error,

$$L^{\mathrm{val}}_{m,d,s} = \frac{1}{|\mathcal{V}_d|} \sum_{i \in \mathcal{V}_d} \left(\frac{\widehat{f}_{m,d,s}(x_i) - y_i}{y_i} \right)^2 . \tag{31}$$

Exact ties favor the smaller simplified expression, then the earlier seed in the frozen list. We add no denominator offset: if a validation target is exactly zero, this loss is undefined and that seed is ineligible. One short-panel equation, `feynman-iii.15.12`, has exact-zero clean validation targets, so no Track B seed can be selected for it under this rule. We retain it in the declared panel and distinguish this selector limitation from a method's failure. Neither the test targets nor the true expression participate in selection. The fixed search configuration has no per-equation outer tuning and a one-hour fitting/search cap; symbolic evaluation has separate, recorded time allowances.

Track C: joint corruption. The declared Track C protocol uses the same equations, official rows, method settings and five algorithm seeds as Track B, with one immutable corrupted version of each equation shared by all methods and search seeds. The current collected rows do not pass that shared-input check; Section N.3 reports the resulting provisional outcomes and the input-identity discrepancy. The primary condition adds independent Gaussian target noise to training and validation targets with standard deviation

$$\sigma_d = 10^{-2} \sqrt{\frac{1}{|\mathcal{T}_d|} \sum_{i \in \mathcal{T}_d} (y_i^{\mathrm{clean}})^2}, \tag{32}$$

and inserts two irrelevant feature columns using SRSD's random-position and log-uniform-magnitude dummy procedure. We use a separate fixed corruption seed, 2027, and record the sampled column positions and scales. This is a custom *combined* condition: the noise level and dummy count

do not vary within the main comparison. The code also supports zero corruption and other SRSD-style levels for separate extensions; those are not pooled into this condition. Validation selection uses Eq. (31) with the *noisy* validation target. Primary test accuracy uses the original clean latent test target, while noisy-target accuracy is a diagnostic. The ground-truth equation is reindexed for the inserted columns before structural scoring. Because both disturbances occur together, the B–C difference does not identify the separate effect of either one. For the zero-target equation noted above, a B–C change can also reflect restored validation-loss eligibility when noise moves those targets away from zero.

L.2 Methods, evaluation and analysis

The prespecified main comparison uses C-KAN with GSR extraction, one-hidden-layer PyKAN with GSR, PySR (Cranmer, 2023), Operon (Burlacu et al., 2020) and TPSR (Shojaee et al., 2023). Tracks B and C additionally include KAN-SR (Bühler & Guillén-Gosálbez, 2025); its incomplete Track A attempts are retained for audit but excluded from that track's main inferential roster. C-KAN's frozen `ckan_k5` configuration has a population of 150, at most 100 generations, a 1,024-example supervised batch per generation and five extraction candidates. One evolution produces separate vanilla and GSR equations under a shared one-hour fit deadline; GSR, the primary mode, is extracted first, and vanilla extraction uses whatever budget remains. PyKAN's vanilla and GSR outcomes instead come from separately scheduled fits; the two-hidden-layer architecture and vanilla equations are descriptive comparisons. We preserve each baseline's frozen native search and checkpoint vocabulary rather than asserting identical operator implementations. Method-specific transformations are fitted on training data and exported equations are checked in original feature and target coordinates. Jobs are capped at 10 GB and one CPU thread, with one GPU for methods using their pinned GPU implementation; host, device and package identities are recorded for each attempt.

A returned expression must yield complete finite holdout predictions and agree with native predictions within the frozen tolerance to count as fully evaluable. Track A reports BB run-level and best-of-ten clipped-R^2 performance profiles, FP held-out R^2, expression-tree size, completion and controlled runtime. The best-of-ten profile is a search-potential diagnostic, not expected single-run performance. Compatible published PySR and Operon SRBench records can contribute to accuracy comparisons; where raw predictions are unavailable, their validity is labelled provisional. Runtime comparisons use local runs from five frozen seeds on a designated timing cohort, not heterogeneous public times.

For Tracks B and C, we apply the pinned SRSD canonicalization and unit-cost tree-edit comparator to each selected expression and its ground truth. Normalized edit distance is capped at one by the ground-truth tree size; $\mathrm{NED} = 0$ denotes skeleton recovery under this comparator, not a proof of algebraic equivalence. Runs that fail full evaluation score NED 1 and recovery 0; missing attempts remain missing until collected. We report validation-selected recovery, mean scored NED, the fraction with clean-test $R^2 > 0.999$, joint structural-and-numerical recovery, evaluability and conditional expression size. Run-level recovery and any-seed recovery are descriptive; the latter uses an oracle over seeds. Track C additionally reports dummy-variable use among evaluable selected expressions and paired changes from Track B. Overall short-panel rates are difficulty-weighted by the full SRSD proportions (30/40/50), while difficulty-specific outcomes remain visible; weighting does not make the curated panel a full-benchmark estimate.

Statistical tests use datasets as paired units, never individual algorithm seeds. Track A uses Friedman omnibus tests followed, when warranted, by paired two-sided Wilcoxon comparisons against C-KAN with Holm correction within each prespecified accuracy or size family. Tracks B and C use Cochran's Q, then exact paired McNemar tests for binary skeleton recovery, and Friedman followed by paired Wilcoxon tests for scored NED; the recovery and NED families receive separate Holm corrections. Stratified paired bootstrap intervals resample equations within difficulty, using common indices across methods and, for B–C deltas, across tracks. We interpret cross-machine runtime descriptively, since CPU and GPU methods run on their intended hardware. Any B–C comparison collected with different device classes is flagged rather than attributed solely to corruption. Protocol hashes bind source, data, budgets, environment and checkpoints; failed and interrupted attempts remain in append-only ledgers rather than disappearing from denominators.

M BENCHMARK DEFINITIONS AND EVALUATION PROTOCOL

M.1 FROZEN MAIN-PAPER ROSTERS

Table 9 lists the six BB and six FP SRBench datasets in the main-paper profile. The extended roster has twelve datasets in each family. Track A uses the first ten official seeds: $23654, 15795, 860, 5390, 16850, 29910, 4426, 21962, 14423, 28020$. The controlled timing subset uses the first five of these. The extended profile supports all thirty official seeds.

Table 9: Frozen Track A datasets.

Black-box	First-principles
1028_SWD	first_principles_absorption
1089_USCrime	first_principles_ideal_gas
1199_BNG_echoMonths	first_principles_kepler
192_vineyard	first_principles_rydberg
522_pm10	first_principles_schechter
650_fri_c0_500_50	first_principles_supernovae_zr

Table 10 lists the identical thirty-equation roster used in Tracks B and C. Both tracks use algorithm seeds $23654, 15795, 860, 5390, 16850$. The full SRSD-Feynman roster has 30 Easy, 40 Medium and 50 Hard equations and supports ten fixed seeds.

Table 10: Frozen SRSD-Feynman equations for Tracks B and C.

Easy	Medium	Hard
feynman-i.12.1	feynman-i.8.14	feynman-i.6.20a
feynman-i.14.4	feynman-i.11.19	feynman-i.30.3
feynman-i.18.16	feynman-i.12.11	feynman-i.44.4
feynman-i.25.13	feynman-i.16.6	feynman-ii.13.23
feynman-i.27.6	feynman-i.18.4	feynman-ii.35.21
feynman-i.47.23	feynman-i.29.4	feynman-iii.9.52
feynman-ii.2.42	feynman-ii.6.11	feynman-bonus.6
feynman-ii.8.31	feynman-iii.4.32	feynman-bonus.12
feynman-ii.15.4	feynman-iii.15.12	feynman-bonus.17
feynman-ii.38.14	feynman-bonus.18	feynman-bonus.20

M.2 EVALUATION DETAILS

Track A requires complete finite test predictions and a parseable equation. Failures contribute zero clipped R^2 to BB profiles, while missing raw scores and sizes remain missing. Size counts SymPy preorder nodes without full simplification. Published observations without raw predictions are provisional. Datasets, not seeds, are the statistical units.

For Tracks B and C, the pinned SRSD canonicalizer replaces numerical constants with one tree label and retains variable names. Its unit-cost tree edit distance (TED) yields

$$\mathrm{NED}(\widehat{f}, f^*) = \frac{\min\{\mathrm{TED}(T(\widehat{f}), T(f^*)),\ |T(f^*)|\}}{|T(f^*)|}.$$

NED zero denotes a syntactic skeleton match; invalid or timed-out expressions score NED one. Selection eligibility uses only finite validation evidence, then breaks ties by simplified size and seed order. In the clean validation file for feynman-iii.15.12, 873 of 1,000 targets are exactly zero; pointwise relative validation loss is therefore undefined for every method and seed. This is recorded as a selection limitation, not an observed symbolic-search failure. Track C holds dummy positions and scale/sign settings fixed across splits, draws new dummy values and Gaussian noise for each split, and keeps corruption randomness independent of the method seed. Manifests record file identities and variable reindexing.

Table 11: Exploratory Track A results; controlled timing remains blank

Method	BB AUC-run ↑	BB AUC-best ↑	FP R^2 ↑	Size ↓	Valid ↑	Time (s) ↓
C-KAN + GSR	0.332	0.476	0.922 [0.853,0.989]	30.0 [27.8,55.5]	86.7%	—
C-KAN vanilla	0.276	0.437	0.887 [0.847,0.967]	51.5 [27.5,68.2]	86.7%	—
PyKAN-1 vanilla	0.249	0.448	-2.026 [-9.966,0.070]	157.0 [112.8,369.5]	80.0%	—
PyKAN-2 vanilla	0.149	0.297	-11.460 [-134.215,-0.061]	1178.2 [755.4,3089.8]	73.3%	—
PySR	0.435	0.572	0.984 [0.931,0.991]	22.2 [12.4,26.6]	100.0%	—
Operon	0.435	0.603	0.936 [0.842,0.992]	16.8 [10.0,35.8]	100.0%	—
TPSR	0.179	0.376	0.956 [0.849,0.971]	32.2 [23.4,41.9]	100.0%	—

12 datasets, 10 seeds per method. FP R^2, size, and time are median [Q1,Q3] across datasets. Public-archive validity is provisional because raw predictions are unavailable.

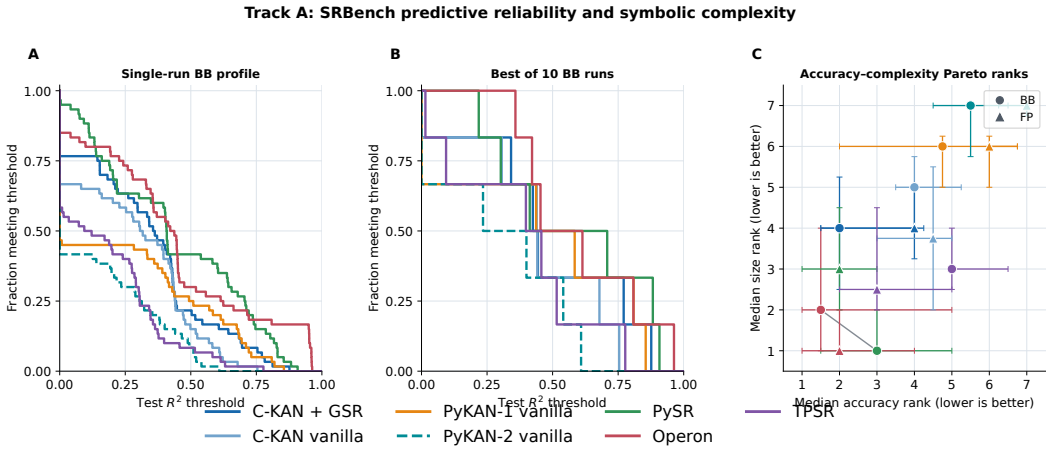

Figure 3: Track A on six black-box (BB) and six first-principles (FP) SRBench datasets. (A) The fraction of BB dataset–seed runs reaching a clipped test-R^2 threshold; a failed run contributes zero. Area under the curve is AUC-run. (B) The best score among ten BB seeds per dataset, a search-potential diagnostic rather than deployable seed selection. (C) Median dataset-level accuracy and valid-expression-size ranks with equation-bootstrap intervals; lower ranks are better. The Pareto front is descriptive and uses the revised seven-outcome roster.

N Detailed benchmark results

This appendix gives the full per-track analyses summarized in Section 4: evidence status, intervals, exploratory tests, failure modes and the complete ablation discussion.

N.1 Track A: predictive performance on SRBench

Evidence and comparison. The twelve-dataset short panel contains ten selected production seeds per dataset for C-KAN-GSR, C-KAN-Vanilla, PyKAN-1-Vanilla, PyKAN-2-Vanilla, PySR, Operon and TPSR. The two C-KAN rows are extractions from shared evolutions; the PyKAN rows are separate fixed one- and two-hidden-layer networks. For PyKAN we use the complete standalone-vanilla protocol and exclude nine conflicting earlier-version keys. KAN-SR was discontinued on Track A. This seven-outcome roster replaces the earlier PyKAN-1-GSR comparison *after* results were inspected, so statistical comparisons are exploratory. C-KAN and PyKAN scores have local prediction evidence. Most accepted PySR, Operon and TPSR rows come from the published SRBench archive: their expressions were parsed, but raw predictions were unavailable, so predictive validity is provisional. Four PyKAN-2 first-principles attempts report a software `UnboundLocalError`; the protocol scores them as failures, but they should not be read as evidence of an inherent architectural limit. Controlled timing cohorts are still incomplete.

Black-box prediction. C-KAN-GSR has BB AUC-run 0.332, and the paired vanilla extraction has 0.276. Both exceed PyKAN-1-Vanilla (0.249) and PyKAN-2-Vanilla (0.149), but trail PySR

and Operon (both 0.435); TPSR has 0.179. The best-of-ten AUCs are 0.476 and 0.437 for the two C-KAN modes, 0.448 and 0.297 for PyKAN-1 and PyKAN-2, 0.572 for PySR, 0.603 for Operon and 0.376 for TPSR (Figure 3). C-KAN-GSR beats PySR's run-level score on four of six BB datasets, but all ten C-KAN seeds on `650_fri_c0_500_50` exhaust the fit deadline without an equation, reversing the aggregate ordering. This 50-feature dataset gives C-KAN's minimal fully connected input-to-output genome 50 initial spline edges. That makes search cost a plausible explanation, although we did not profile this run sufficiently to establish causation. PySR also leads C-KAN-GSR on `1089_USCrime` (0.712 versus 0.508). In the revised accuracy–complexity rank panel, only PySR and Operon lie on the BB Pareto front.

First-principles prediction and equation size. Across the six FP datasets, the median of failure-penalized dataset test-R^2 values is 0.922 for C-KAN-GSR and 0.887 for C-KAN-Vanilla. PyKAN-1-Vanilla gives -2.026 and PyKAN-2-Vanilla -11.460, whereas PySR, Operon and TPSR give 0.984, 0.936 and 0.956. The negative PyKAN summaries reflect very poor selected fits and the declared failure penalty; neither PyKAN topology returns a valid equation for Kepler. C-KAN-GSR attains 0.983 on absorption and 0.999 on supernovae, but only 0.643 on Kepler. Its median conditional expression size is 30.0 SymPy-tree nodes, compared with 51.5 for C-KAN-Vanilla, 157.0 for PyKAN-1 and 1,178.2 for PyKAN-2. The PyKAN size medians cover only the ten datasets on which each returns at least one valid equation; both C-KAN medians cover eleven. PySR and Operon remain smaller at 22.2 and 16.8 nodes over all twelve. Size therefore needs to be read alongside reliability, and neither C-KAN mode reaches the FP accuracy–size Pareto front.

Reliability and inference. C-KAN-GSR and C-KAN-Vanilla each return 104 valid equations in 120 runs (86.7%); PyKAN-1 and PyKAN-2 return 96 (80.0%) and 88 (73.3%), respectively. The recorded PySR, Operon and TPSR rates are 120/120, mostly under provisional published-evidence rules. Each C-KAN mode has 16 invalid outcomes; twelve dataset–seed failures overlap, including all ten `650_fri_c0_500_50` timeouts. GSR yields higher BB AUC-run, higher FP median R^2, and smaller equations than vanilla in this configured pipeline. Because GSR runs first and vanilla receives the remaining shared deadline, this is not an isolated comparison of extraction backends.

With the revised roster, exploratory Friedman tests detect differences in BB accuracy, FP accuracy and size, but none of C-KAN-GSR's Holm-adjusted BB or FP paired comparisons passes 0.05 on six datasets per family (Appendix O). The size comparison with PySR favors PySR after Holm adjustment. No method has a complete five-seed serial timing cohort, so Table 11 leaves controlled runtime blank. The descriptive result is that C-KAN is stronger than either tested fixed PyKAN topology on this panel, while PySR and Operon remain stronger on aggregate BB reliability and produce smaller equations.

N.2 Track B results: recovery from clean data

Evidence and budget status. All eight outcomes in the revised comparison have complete five-seed blocks on the 30 short-panel equations: C-KAN-GSR, C-KAN-Vanilla, PyKAN-1-Vanilla, PyKAN-2-Vanilla, KAN-SR, PySR, Operon and TPSR. The C-KAN modes share each evolution; the PyKAN topologies are trained separately. The C-KAN main protocol has a 3,600-second fit allowance. The standalone PyKAN vanilla files declare 5,000 seconds, a protocol deviation, but all 300 selected terminal runs finished before 3,600 seconds (one additional interrupted attempt is excluded; maximum selected elapsed time under 810 seconds). The analyzer admits only these nonbinding-budget attempts and retains the discrepancy in its audit; it does not rewrite the raw observations. Earlier paired PyKAN-1 runs belong to incompatible, incomplete protocols and are excluded. All eight analyzed methods used the same official clean SRSD inputs. Because the comparison roster was revised after seeing results, and the PyKAN timeout configuration differed, the paired tests in Appendix P are exploratory rather than prespecified confirmatory evidence. PyKAN-GSR remains incomplete and is not assigned a main-table row.

Structural recovery. PySR recovers the most skeletons: 15 of 30 equations, or 43.3% (95% interval 31.7–55.0%) after weighting the Easy, Medium and Hard groups in SRSD's proportions. All of its recoveries are Easy (8/10) or Medium (7/10), and it recovers none of the Hard equations (Figure 4A). KAN-SR recovers five equations, including one Hard one (15.8%, 4.2–30.0%). Its skeleton recovery is therefore well below PySR's, although its high-accuracy rate is slightly higher

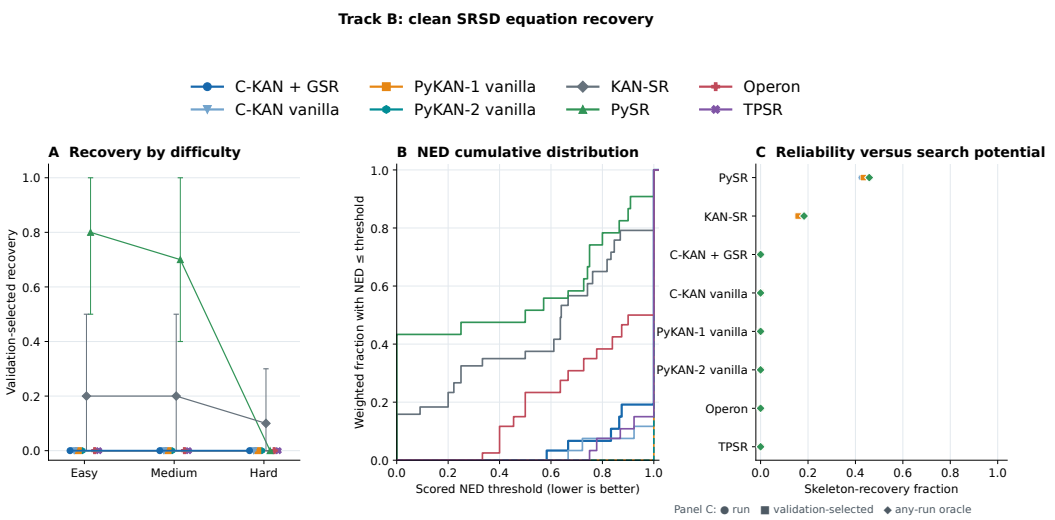

Figure 4: Track B outcomes on the clean 30-equation SRSD-Feynman panel. (A) Validation-selected skeleton recovery by difficulty, with 95% stratified equation-bootstrap intervals. (B) Difficulty-weighted empirical CDF of scored NED. Failed or ineligible selections score NED 1, which is why every curve jumps at the right edge. (C) Run-level (●), validation-selected (■) and any-seed oracle (⧫) recovery. The oracle is a search-potential diagnostic, not deployable performance. Both C-KAN modes and both standalone vanilla PyKAN topologies are shown; incomplete PyKAN-GSR runs are excluded.

Table 12: Exploratory Track B clean SRSD equation recovery

Method	SR_E^{val}	SR_M^{val}	SR_H^{val}	SR_A^{val}	NED_E	NED_M	NED_H	NED_A	$R^2 > .999$	Joint	Size ratio	Valid runs
C-KAN + GSR	0.0%	0.0%	0.0%	0.0%	1.000	0.925	0.957	0.957	37.5%	0.0%	7.50 [0.67,15.00]	50.3%
C-KAN vanilla	0.0%	0.0%	0.0%	0.0%	1.000	0.967	0.965	0.974	25.0%	0.0%	3.29 [1.58,8.44]	37.0%
PyKAN-1 vanilla	0.0%	0.0%	0.0%	0.0%	1.000	1.000	1.000	1.000	0.0%	0.0%	14.60 [8.00,35.75]	37.3%
PyKAN-2 vanilla	0.0%	0.0%	0.0%	0.0%	1.000	1.000	1.000	1.000	0.0%	0.0%	44.64 [44.64,44.64]	1.3%
KAN-SR	20.0%	20.0%	10.0%	15.8%	0.341	0.564	0.706	0.567	77.5%	15.8%	1.04 [0.90,1.30]	91.3%
PySR	80.0%	70.0%	0.0%	43.3%	0.167	0.275	0.702	0.426	71.7%	43.3%	1.00 [0.62,1.19]	96.7%
Operon	0.0%	0.0%	0.0%	0.0%	0.713	0.830	0.848	0.808	55.0%	0.0%	1.50 [1.20,1.67]	62.7%
TPSR	0.0%	0.0%	0.0%	0.0%	1.000	0.962	0.970	0.975	0.0%	0.0%	3.20 [1.16,4.27]	85.0%

30 equations, 5 seeds per method–equation. All SR values are validation-selected unless marked otherwise. Overall rates and mean NED use the original SRSD difficulty weights $30/120, 40/120, 50/120$. Size ratio is the weighted median [Q1,Q3] among fully evaluable selected equations; validity is the fraction of terminal runs with a fully evaluable equation.

(77.5% versus 71.7%). C-KAN-GSR, C-KAN-Vanilla, both PyKAN topologies, Operon and TPSR recover no skeleton. For C-KAN this is not an artifact of validation-based seed selection. None of its 150 GSR runs and none of its 150 vanilla runs reached NED 0, so the any-seed oracle is also zero (Figure 4C). One Medium equation, `feynman-iii.15.12`, has no eligible seed for any method, because its clean validation targets are exactly zero (§L.1).

Mean scored NED gives a finer ordering: PySR 0.426 (0.316–0.543), KAN-SR 0.567, Operon 0.808, C-KAN-GSR 0.957 (0.919–0.989), C-KAN-Vanilla 0.974, TPSR 0.975, and both PyKAN topologies 1.000. The CDF in Figure 4B shows why the C-KAN mean is so close to one. Only about a fifth of its difficulty-weighted selections have NED below one, and the lowest is 0.583. On this clean panel, therefore, C-KAN does not reproduce the ground-truth structure. The result does not support the claim that constructive search by itself yields better structural recovery than fixed-template KAN or genetic-programming baselines.

Accuracy without structure. NED and prediction accuracy separate sharply, and not only for C-KAN. Operon reaches clean-test $R^2 > 0.999$ on 55.0% of the weighted panel without a single exact skeleton. C-KAN-GSR reaches it on 12 of 30 equations (37.5%, 20.8–55.0%), and C-KAN-Vanilla on 8 (25.0%); neither PyKAN topology reaches it on a validation-selected equation. The structurally evaluable C-KAN equations are often large, although this size summary is not restricted to accurate

fits. Among its 13 fully evaluable selections, the median ratio of simplified size to ground-truth size is 7.5 (interquartile range 0.67–15). The corresponding medians are 1.0 for PySR, 1.04 for KAN-SR and 1.5 for Operon. PyKAN-1 has a size ratio of 14.6 among 15 structurally evaluable selections; PyKAN-2 has a ratio of 44.6 from only one, too little for a stable size summary. The per-equation outcomes (Appendix P) show three recurring patterns:

- **Correct term plus residue.** On `feynman-ii.8.31` ($u = \frac{1}{2}\epsilon_0 E^2$), C-KAN returns $4.427 \times 10^{-12}E^2 - 6.2 \times 10^{-17}E + 2.2 \times 10^{-14}$. The leading term is correct and the test R^2 is 1.000, but the two negligible terms make the tree edit distance exceed the ground-truth size, so NED is capped at 1. On `feynman-i.29.4` ($k = \omega/c$), an additive offset of 8.3×10^{-7} likewise leaves NED at 0.667.
- **Faithful but unsimplified products.** On `feynman-i.12.1` ($F = \mu N$), the selected equation fits to $R^2 = 1.000$. It represents the product indirectly. A two-input geometric node, Eq. (2) with $\mu_v = \frac{1}{2}$, computes a smoothed signed square root of the product of two affine edges, and a downstream squaring edge restores the product. In C-KAN's internal coordinates this equation has 41 nodes. After the input and target scalings are removed analytically and the result is simplified, the $\epsilon_\times$ term blocks cancellation of the root, and the equation grows to 252 nodes instead of 3.
- **Physically meaningful approximations.** On relativistic velocity addition, `feynman-i.16.6`, the selected equation is approximately the Galilean limit, $0.99(u+v)$ plus an offset ($R^2 = 0.9994$, NED 0.583). This is accurate on the sampled range but structurally different from the target.

In the first pattern, the unit-cost NED counts an artifact of the network parameterization, a negligible residual term, as structural error. The post-hoc clean-up test below shows that this explanation covers only a small part of C-KAN's gap. The other two patterns are genuine structural differences.

Where the equation is lost. C-KAN records each stage of its pipeline, which lets us separate the constructive search from symbolic conversion and export. The telemetry scores networks and candidates on C-KAN's internal 10% training-only split with standardized targets, so this diagnostic never touches official labels. We call a model numerically accurate when its NMSE on that split is below 10^{-3}. No pre-symbolic score on the official test set was recorded, so this internal measure is the only pre-symbolic accuracy available.

By this measure, the final-generation champion is accurate in at least one seed for 9 of the 30 equations. After pruning and refitting, the best prepared network is accurate for 13 (in 46 of 149 runs). After symbolic conversion, the best candidate is accurate for 12, ten of which overlap with the network count. At the run level, 39 of the 46 runs with an accurate prepared network also produce an accurate symbolic candidate. Conversion therefore loses little accuracy, and the pruning-and-refit stage improves on the evolved champion. The ceiling is set by the search itself. It finds no accurate network in any seed for 17 equations, including the multi-product sum `feynman-i.11.19` (selected test $R^2 = 0.42$) and `feynman-ii.6.11` (0.17).

Table 21 assigns every equation to the first stage at which C-KAN-GSR loses it, using a fixed rule over its five seeds. Search accounts for most losses, especially at Medium and Hard difficulty. Of the ten equations whose networks were accurate, three lose accuracy at conversion. Seven keep an accurate symbolic candidate but are not recovered even after the post-hoc clean-up below.

efsec:shortfall.

Is the structural loss cosmetic? A post-hoc clean-up test. The residue pattern raises an obvious question: would ordinary symbolic post-processing close the gap? After seeing the results, we applied KAN-SR's own clean-up rules (Bühler & Guillén-Gosálbez, 2025) to every returned equation of every Track B method, using training data only:

- *Rounding and refit:* every constant is rounded to two decimals in KAN-SR's scaled coordinates, and the surviving constants are refit by least squares.
- *Term elimination:* KAN-SR's optional backward elimination, which removes a constant when training error rises by at most 1%.

We then re-scored each equation with the benchmark's own canonicalizer, tree distance, validation-based seed selection and bootstrap (Appendix R). For C-KAN-GSR, validation-selected recovery rises from 0 to 5.8% (95% interval 0–15.0%). Two equations are recovered, `ii.8.31` and `i.29.4`, both from the residue pattern. Mean NED falls from 0.957 to at best 0.873, and the high-accuracy rate moves from 37.5% to 41–45%.

The same procedure changes the baselines more. Operon rises from 0 to 24.2%: its equations were correct up to offsets of order 10^{-9}. KAN-SR rises from 15.8% to 30.8%, and TPSR from 0 to 9.2%. PySR falls from 43.3% to 27.5%. The canonicalizer removes a constant only when it is exactly one, so rescaling and refitting turns an exact x_0x_1 into $1.0000001\,x_0x_1$, which no longer matches.

Two conclusions follow:

- *Residue explains little of C-KAN's gap.* With the same post-processing for every method, C-KAN still has the lowest recovery.
- *Differences of a few points in skeleton recovery are fragile.* The metric is sensitive to constant formatting in both directions, so we keep the prespecified, unprocessed results as primary.

C-KAN's remaining accurate but unrecovered equations are structural:

- *Geometric-node export:* on `i.12.1`, the cleaned equation is $x_0^2x_1^2/(x_0^2x_1^2)^{0.5}$. This equals x_0x_1 for the positive SRSD inputs, but the canonicalizer does not assume positivity.
- *Aliased primitives:* on `ii.15.4`, the result is $-x_0x_1\sin(x_2 - 3\pi/2)$, which is exactly $-x_0x_1\cos x_2$.
- *Approximating primitives:* saturating or oscillating primitives with extreme affine constants stand in for simpler terms, or a cubic polynomial stands in for a square root.

Reliability, budget and extraction mode. The selected C-KAN-GSR equation is fully evaluable on 13 of 30 equations; the two PyKAN variants have 15/30 and 1/30 fully evaluable selections. At the run level, 76 of 150 C-KAN-GSR runs are fully evaluable (50.3% after difficulty weighting), against 37.3% for PyKAN-1, 1.3% for PyKAN-2, 91.3% for KAN-SR and 96.7% for PySR. The dominant failure is not an invalid equation. In 58 GSR runs, the SRSD canonicalizer exceeded its 120-second allowance. These expressions are about three times longer than C-KAN's evaluable ones (median 704 versus 241 characters), and 71% of them contain a fractional power or root, against 16% of the rest. Five of C-KAN-GSR's twelve high-accuracy selections fail in this way, and they receive NED 1 under the prespecified rule. The remaining terminal outcomes are 7 extraction timeouts without a valid candidate, 5 constant-only expressions that cannot enter validation selection, 1 otherwise finite expression on the exact-zero-target equation (whose pointwise validation loss is undefined), 2 supervisor timeouts and 1 canonicalizer error.

All 149 C-KAN-GSR runs with generation telemetry completed 100 generations. Evolution took a median of 2,200 seconds of the 3,600-second budget, preparation (pruning and refitting) took 137 seconds, and GSR extraction took 571 seconds. Vanilla extraction is fast (median 10 seconds), but it runs after GSR under the shared deadline, and 57 vanilla runs found no time left for a valid candidate. The lower vanilla validity (37.0% of runs) and accuracy therefore reflect the mode ordering as much as the backends themselves, and the two modes should not be read as a controlled comparison of extractors. The PyKAN failure pattern is different: the SRSD canonicalizer times out in 79/150 PyKAN-1 and 143/150 PyKAN-2 standalone-vanilla runs on Track B. This is a failure of the complete equation-export-and-scoring pipeline; it does not, by itself, prove that the trained spline networks could not predict. Median elapsed time is 3,530 seconds for C-KAN, 3,057 for PySR and under 320 for KAN-SR, TPSR and Operon. These times come from heterogeneous machines and are descriptive only.

Revised comparison and inference. The exploratory equation-level tests distinguish PySR's skeleton recovery from C-KAN-GSR's (Holm-adjusted exact McNemar $p < 0.001$). They find no skeleton-recovery difference between C-KAN and either PyKAN topology because all three are at zero. The NED comparison with PyKAN is likewise not significant after Holm correction: when most selections receive the ceiling penalty of one, a lower mean C-KAN NED need not create a

Table 13: Provisional Track C: jointly corrupted SRSD equation recovery

Method	SR_C^{val}	D^{SR} (pp)	Mean NED$_C$	D^{NED}	$R^2 > 0.999$	Joint	Dummy use	Fully eval.
C-KAN-GSR	0.0%	+0.0	0.966	+0.009	35.8%	0.0%	7.7% (1/16)	54.2%
C-KAN-Vanilla	0.0%	+0.0	0.991	+0.016	20.0%	0.0%	7.2% (1/17)	57.5%
PyKAN-1-Vanilla	0.0%	+0.0	1.000	+0.000	0.0%	0.0%	18.3% (3/15)	50.0%
PyKAN-2-Vanilla	0.0%	+0.0	1.000	+0.000	0.0%	0.0%	100.0% (2/2)	8.3%
KAN-SR	8.3%	+7.5	0.739	+0.172	57.5%	8.3%	4.2% (1/30)	100.0%
PySR	40.0%	+3.3	0.407	-0.019	70.8%	40.0%	4.2% (1/30)	100.0%
Operon	0.0%	+0.0	0.806	-0.003	55.0%	0.0%	15.3% (3/25)	81.7%
TPSR	0.0%	+0.0	0.970	-0.005	0.0%	0.0%	78.6% (17/22)	70.0%

30 equations, 5 algorithm seeds per method. Overall rates and mean NED use the original SRSD difficulty weights $30/120, 40/120, 50/120$. $D^{SR} = SR_B^{\text{val}} - SR_C^{\text{val}}$; $D^{NED} = \overline{NED}_C - \overline{NED}_B$. Positive degradation means worse recovery under corruption. Dummy-use counts are raw numerators/denominators among structurally evaluable selected equations; percentages are difficulty-weighted. C-KAN-Vanilla and C-KAN-GSR share evolution and are separate extraction outcomes, not independent runs.

large paired rank difference. These tests are useful diagnostics, but the revised roster and PyKAN budget deviation preclude presenting them as prespecified confirmation (Appendix P).

Summary. On clean SRSD data, PySR is the strongest recovery method, and KAN-SR is the only KAN-based method with exact skeletons. Both C-KAN modes exceed the fixed PyKAN topologies in selected high-accuracy rate. C-KAN-GSR also yields more fully evaluable runs than either PyKAN topology, whereas C-KAN-Vanilla and PyKAN-1 have similar run-level evaluability; none recovers a skeleton. C-KAN's main limit remains the constructive search, which finds no accurate prepared network for 17 of 30 equations; where it succeeds, export can produce long or structurally different formulas. The deeper PyKAN topology is particularly hard for the canonicalizer to process. Post-hoc clean-up recovers two C-KAN equations and helps some baselines more. Appendix P reports difficulty intervals, run-level and oracle recovery, exploratory tests and every selected equation.

Incomplete PyKAN-GSR pilot. GSR is a borrowed symbolic backend (Sovrano et al., 2026). In an earlier, incompatible 42-pair PyKAN-1 pilot, GSR yielded test $R^2 > 0.999$ in 22 attempts versus none for the paired vanilla outputs, but neither mode yielded a structurally evaluable equation. This subset cannot replace the complete standalone-vanilla comparison or support a Track B recovery claim for PyKAN-GSR.

N.3 Track C: recovery under joint corruption

Evidence status. The present Track C ledger has complete five-seed blocks on all 30 declared equations for C-KAN-GSR, C-KAN-Vanilla, PyKAN-1-Vanilla, PyKAN-2-Vanilla, KAN-SR, PySR, Operon and TPSR. The two C-KAN rows share one evolved population per seed; the PyKAN topologies are trained separately. PyKAN-GSR remains incomplete and is excluded. The paired Track B PyKAN-vanilla files declared a 5,000-second fit limit, although every selected terminal attempt finished before 3,600 seconds; this nonbinding budget difference is disclosed in §N.2. More seriously, the corrupted-array hashes differ across collection machines for every Track C equation, even though each method reused one fixed array across its own five seeds. Thus the literal shared-corruption condition in §L.1 is unverified. The paired *clean* dataset identities match Track B, but the results below remain provisional, method-wise descriptions. Between-method contrasts and B–C changes are observations, not controlled corruption effects. The revised comparison roster is exploratory, and the planned inferential tests remain withheld pending the corrupted-input audit.

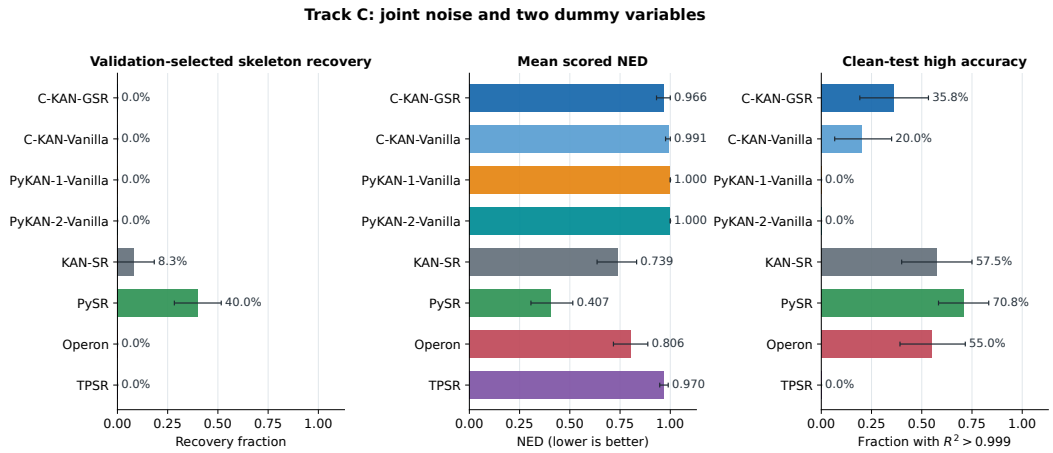

Figure 5: Provisional Track C outcomes on the 30-equation short panel. Bars show difficulty-weighted validation-selected skeleton recovery, mean scored normalized edit distance (NED), and the fraction with clean-test $R^2 > 0.999$; whiskers show stratified equation-bootstrap 95% intervals within each method. A failed or ineligible selected equation scores NED 1 and recovery 0. Both standalone vanilla PyKAN topologies are included. Cross-method corrupted-input hashes currently disagree, so the display is descriptive.

Structural recovery and numerical accuracy. PySR has the highest recorded validation-selected skeleton recovery: 14 of 30 equations, or 40.0% after weighting the Easy, Medium and Hard groups in the original SRSD proportions. Its mean scored NED is 0.407. KAN-SR recovers three equations (8.3% weighted; mean NED 0.739). Neither C-KAN extraction mode, either PyKAN topology, Operon nor TPSR achieves an exact skeleton in these recorded outcomes. C-KAN-GSR has mean NED 0.966, compared with 0.991 for C-KAN-Vanilla; both PyKAN topologies score 1.000, while Operon and TPSR have 0.806 and 0.970. These are scored values, so extraction and evaluation failures contribute the prespecified penalty rather than disappearing from the mean.

Prediction and structural recovery tell different stories. The clean-test $R^2 > 0.999$ rate is 70.8% for PySR, 57.5% for KAN-SR and 55.0% for Operon, although Operon has no exact recoveries. C-KAN-GSR reaches 35.8% (11 of 30 equations) and C-KAN-Vanilla 20.0% (6 of 30); neither PyKAN topology nor TPSR reaches this threshold on a validation-selected equation. GSR has higher recorded numerical accuracy and modestly lower scored NED than vanilla extraction from shared evolution, but neither mode produces an exact skeleton in this panel. GSR runs first under their shared deadline, so this is an observed pipeline comparison rather than an isolated test of the two extractors. The observed pattern does not support claiming that C-KAN is the strongest symbolic-recovery method in Track C. Nor is corruption what prevents C-KAN's recovery. It recovers no skeleton on the clean Track B panel either, where most losses occur in the search itself and the rest at export (§N.2). The post-hoc clean-up of §N.2, applied to C-KAN's Track C equations, recovers one equation, `i.29.4` (3.3%, interval 0–10%; Appendix R). The baselines' corrupted inputs are not stored locally, so they could not be re-scored in the same way.

Difficulty and irrelevant-variable use. The recorded PySR recovery is concentrated in Easy (8/10) and Medium (6/10) equations; it recovers none of the Hard ten. Its mean scored NED rises from 0.111 through 0.330 to 0.646 across those groups. KAN-SR recovers two Easy and one Medium equation, with no Hard recovery. Neither C-KAN mode nor either PyKAN topology recovers a skeleton at any difficulty. Both PyKAN topologies have scored NED 1 in every difficulty group; this ceiling combines structural differences with ineligible or failed symbolic evaluations. This unevenness is a reason to retain difficulty-stratified results alongside the weighted overall estimate (Appendix Q).

The dummy-use diagnostic asks whether at least one inserted irrelevant symbol remains in the *simplified, validation-selected* expression. It is conditional on structural evaluability, not a failure rate over all 30 equations. One of 16 evaluable C-KAN-GSR equations and one of 17 evaluable C-KAN-

Vanilla equations retain a dummy; the difficulty-weighted conditional rates are 7.7% and 7.2%. The corresponding raw counts are 3/15 for PyKAN-1, 2/2 for PyKAN-2, 1/30 for PySR, 1/30 for KAN-SR, 3/25 for Operon and 17/22 for TPSR. These denominators differ substantially; PyKAN-2 has only two evaluable selections, so its 2/2 dummy-use result is especially unstable. A low conditional dummy-use rate alone is not evidence of better overall recovery. On evaluable selected equations, C-KAN-GSR retains true variables with mean precision 0.969 and recall 0.812 (16 equations); C-KAN-Vanilla gives 0.993 and 0.790 (17 equations). PySR gives 0.989 and 0.897 across 30. PyKAN-1 gives 0.912 and 0.900 on 15 evaluable selections, yet none has the correct skeleton. PyKAN-2 gives 0.417 and 1.000 on only two evaluable selections, too few for a reliable topology comparison. Avoiding dummy symbols can therefore coexist with missed explanatory inputs and a wrong equation tree (Appendix Q).

Clean-to-corrupted changes. All eight displayed outcomes now have complete Track B and Track C seed blocks with matching clean-dataset identities. We report degradation as $D^{SR} = SR_B - SR_C$ and $D^{NED} = \overline{NED}_C - \overline{NED}_B$, so positive values mean worse under corruption. The intervals are paired 95% equation-bootstrap intervals that reuse the same resampled equations in both tracks.

KAN-SR shows the clearest change. Its recovery falls from 15.8% to 8.3% ($D^{SR} = 7.5$ points, interval 0–18.3), and its mean NED rises from 0.567 to 0.739 ($D^{NED} = +0.172$, interval 0.062–0.288). It is the only method whose NED interval excludes zero. Its Track B runs used a GPU and its Track C runs a CPU, however, so this change is not purely a corruption effect. PySR changes little. Its recovery goes from 43.3% to 40.0% (3.3 points, interval -5.0 to 12.5), and its NED moves from 0.426 to 0.407 (interval -0.118 to 0.086). That small NED improvement is an observed change in the selected equations, not evidence that noise helps the method. Operon and TPSR stay at zero recovery in both tracks, and their NED intervals are centred near zero (-0.003 and -0.005). Both PyKAN topologies also remain at zero recovery and scored NED 1 in both tracks. This floor cannot establish that they are robust to corruption; the complete symbolic pipelines did not achieve recovery even on clean data.

C-KAN's recovery is also at the zero floor in both tracks, so this track cannot say whether corruption would damage the skeletons it does not find on clean data. The measurable quantities barely move. C-KAN-GSR's mean NED goes from 0.957 to 0.966 ($+0.009$, interval -0.017 to 0.043), and its high-accuracy rate from 37.5% to 35.8% (12 to 11 equations). C-KAN-Vanilla moves from 0.974 to 0.991 ($+0.016$, interval -0.003 to 0.044) and from 25.0% to 20.0%. The recorded C-KAN differences in numerical fit are small, while structural recovery stays at zero. Because recovery is at a floor and the corrupted inputs disagree across collection machines, these observations cannot establish robustness to the intended one-percent target noise and two irrelevant inputs. Track C combines noise and dummy columns, so none of these differences isolates their individual effects.

Evaluability and inference. A validation-selected equation is fully structurally evaluable on 16/30 C-KAN-GSR and 17/30 C-KAN-Vanilla datasets, 15/30 PyKAN-1 and only 2/30 PyKAN-2 datasets, versus 30/30 for PySR and KAN-SR, 25/30 for Operon and 22/30 for TPSR. On clean Track B the C-KAN counts are 13/30 for both modes, so low evaluability also appears without the Track C corruption. The PyKAN-1 and PyKAN-2 Track C ledgers record 80/150 and 146/150 SRSD canonicalization timeouts, respectively. These dominate their low full-pipeline validity but do not establish that every underlying trained network failed numerically. The C-KAN ledger includes 16 paired fits whose worker exited with code 1, along with fit or extraction timeouts and evaluator-invalid expressions; the recorded worker exit code by itself does not establish the underlying cause. Incomplete evaluability is therefore part of the measured pipeline outcome, not a nuisance to remove after scoring. The Cochran–McNemar and Friedman–Wilcoxon analyses, with Holm correction, remain withheld because the corrupted-input identities disagree across collection machines; the revised eight-outcome roster would in any case be exploratory. Mixed-machine elapsed times appear only as diagnostics in Appendix Q.

N.4 Ablations: full discussion

Design. All ablations use Track B's clean SRSD data, its five seeds, and its validation-based seed selection and scoring. We treat them as two blocks with different equations and nominal budgets.

Within the six-equation block, candidate-count runs also differ in controller version and realized runtime, so those comparisons are descriptive:

- *Component block.* A 2×2 study that removes Safe Mutation (SM), behavior distance, or both. It uses 12 predeclared equations (3 Easy, 4 Medium, 5 Hard), $K = 10$ candidates and a nominal 5,000-second fit cap.
- *Single-change block.* Six equations (1 Medium, 5 Hard). The $K = 5$ and $K = 7$ configurations and the two removals have nominal 3,600-second fit caps; the earlier $K = 10$ pilot has a nominal 5,000-second cap. The older pilot controller sometimes exceeded these caps, so K is not an equal-compute comparison.

Main ($K = 5$) was run twice on those six equations under different code versions. Each comparison uses the Main runs from its own batch: the batch run alongside $K = 7$ for the K study, and the main benchmark batch for the removals. All variants carry the C-KAN 0.3.0 label, but source and protocol identifiers differ across batches.

No variant recovers an exact skeleton on either block, and on the single-change block no variant reaches test $R^2 > 0.999$. The ablations therefore cannot measure effects on recovery. What they can measure is validity, expression size, cost and extraction behavior. With 12 or 6 equations and five seeds, all differences below are descriptive.

efsec:ablations.

Safe Mutation and behavior distance mainly affect validity and cost. In the component block, accuracy differs by at most one equation across the four cells: 2 of 12 equations reach $R^2 > 0.999$ in three cells, and 3 of 12 without either mechanism. Mean scored NED ranges from 0.955 to 1.000, and its mean SM-by-behavior interaction is -0.065. The run-validity interaction is $+6.7$ percentage points, a descriptive contrast on 12 equations rather than evidence of synergy. The clearer differences are in reliability:

- *Validity.* With both mechanisms, 28 of 60 runs return a fully evaluable equation. Without SM, without behavior distance, and without both, the counts are 19, 24 and 19.
- *Canonicalization.* The full configuration has the fewest runs that exceed the canonicalization allowance: 23, against 29–33.
- *Cost.* Among the runs executed on the same hardware, removing SM raised the median fit time from 3,240 to 4,152 seconds.

Species-count trajectories are nearly identical in all four cells (Appendix S); counts alone do not establish whether behavioral diversity changed. The observed differences are more evident in fit and export validity than in accuracy.

The population matters because conversion reorders candidates. Figure 2A shows the global numerical rank of the network that supplies the returned equation. The numerically best network supplies it in only 24%, 17% and 11% of runs for $K = 5$, 7 and 10. The median selected rank is 3–4.5. Figure 2B compares the returned equation with the candidate of lowest symbolic validation loss in the same run:

- In 70 of 81 runs they coincide.
- In the other 11 runs, the simplicity rule saves 5–80% of the expression-tree nodes. Test R^2 falls by at most 0.052 in eight of them. In the remaining three it does not fall: in one it rises by 0.96, because the lowest-loss candidate generalizes poorly.

The value of several candidates therefore comes mostly from symbolic conversion reordering the numerical ranking, not from trading fit for simplicity. On these six equations, none of the observed K batches improved accuracy or recovery; the differing caps and overruns prevent attributing this result to K alone.

Gradient steps and grid extension.

- *Removing gradient steps* lowered the median fit time from 3,059 to 1,262 seconds on the same hardware. On these six equations it lost nothing measurable: 14 of 30 runs were valid against 10, and mean NED was 0.942 against 0.951. Because every variant has zero high-accuracy equations here, this shows only that the local learner's benefit, if any, does not appear on the hardest equations within this budget. It does not show that gradient steps are unnecessary.
- *Removing grid extension* lowered validity to 6 of 30 runs. Its median fit time reached the 3,595-second limit, against 3,059 seconds for Main on the same hardware.

Appendix S reports neural uncertainty, sparsity regularization, the extraction backends across all tracks, and the search dynamics.

O TRACK A PER-DATASET RESULTS AND EXTRACTION DIAGNOSTIC

The short SRBench profile uses six black-box and six first-principles datasets with ten fixed seeds each. The table in this appendix is generated from the checksummed Track A ledger. A black-box accuracy cell is the mean clipped test R^2 across ten runs, with invalid or timed-out runs scored zero; a first-principles accuracy cell is the failure-penalized median test R^2. Size is the median common SymPy-tree count among valid equations. Blank cells mean the method lacks a complete seed block, rather than zero performance. Most PySR, Operon and TPSR rows are accepted published records with provisional prediction validity. The new PyKAN rows come from the complete standalone-vanilla protocol; earlier incompatible attempts were excluded. Four PyKAN-2 FP errors explicitly report a software exception and are scored as failures under the declared rule.

Table 14: Descriptive C-KAN extraction outcomes from shared Track A evolutions. FP R^2 and size are median [Q1,Q3] across datasets. The modes are not independent runs: GSR extracts first, and vanilla uses the remaining fit allowance.

Extraction	BB AUC-run	BB AUC-best	FP R^2	Size	Valid
GSR	0.332	0.476	0.922 [0.853,0.989]	30.0 [27.8,55.5]	104/120
Vanilla	0.276	0.437	0.887 [0.847,0.967]	51.5 [27.5,68.3]	104/120

The aggregate BB profile masks heterogeneous behavior. C-KAN-GSR has a higher run-level AUC than PySR on `1028_SWD`, `1199_BNG_echoMonths`, `192_vineyard` and `522_pm10`. On `650_fri_c0_500_50`, all ten seeds timed out around the one-hour fit limit, so the scored AUC is zero. That dataset accounts for ten of C-KAN-GSR's sixteen invalid outcomes. C-KAN's lower aggregate BB AUC should therefore not be interpreted as uniformly worse prediction on each dataset; nor should its four local wins be interpreted as superior reliability over the selected panel.

The revised seven-outcome dataset-level tests in Table 15 are exploratory because the roster was changed after results were inspected. Holm-adjusted comparisons do not establish a C-KAN-GSR accuracy advantage over any individual comparator on either six-dataset family; the conditional size comparison with PySR favors PySR. Controlled runtime remains unavailable.

Table 15: Exploratory revised Track A dataset-level statistical comparisons.

Endpoint	Baseline	Friedman p	Wilcoxon W	Raw p	Holm p	Median favorable difference
BB accuracy	C-KAN vanilla	0.049	0.0	0.062	0.375	0.047
	PyKAN-1 vanilla		7.0	0.562	1.000	0.150
	PyKAN-2 vanilla		4.0	0.219	1.000	0.276
	PySR		10.0	1.000	1.000	0.048
	Operon		8.0	0.688	1.000	-0.007
	TPSR		4.0	0.219	1.000	0.167
FP accuracy	C-KAN vanilla	< 0.001	4.0	0.219	0.875	0.007
	PyKAN-1 vanilla		0.0	0.031	0.188	2.951
	PyKAN-2 vanilla		0.0	0.031	0.188	12.212
	PySR		4.0	0.219	0.875	-0.030
	Operon		6.0	0.438	0.875	-0.024
	TPSR		10.0	1.000	1.000	0.013
Complexity	C-KAN vanilla	< 0.001	8.5	0.051	0.137	10.750
	PyKAN-1 vanilla		13.0	0.042	0.137	83.750
	PyKAN-2 vanilla		12.0	0.034	0.137	734.500
	PySR		4.0	0.003	0.021	-18.250
	Operon		9.0	0.016	0.081	-20.250
	TPSR		17.0	0.088	0.137	-13.750

Positive paired differences favor C-KAN + GSR. Holm adjustment is separate within each endpoint; interpret pairwise claims only after a significant Friedman test.

Table 16: Exploratory Track A per-dataset results (supplement)

Family	Dataset	Method	Accuracy ↑	Size ↓	Valid
BB	1028_SWD	C-KAN + GSR	0.368	27.0	10/10
BB	1028_SWD	C-KAN vanilla	0.358	62.5	10/10
BB	1028_SWD	PyKAN-1 vanilla	0.374	643.5	10/10
BB	1028_SWD	PyKAN-2 vanilla	0.267	3238.5	10/10
BB	1028_SWD	PySR	0.190	2.0	10/10
BB	1028_SWD	Operon	0.363	20.0	10/10
BB	1028_SWD	TPSR	0.002	40.0	10/10
BB	1089_USCrime	C-KAN + GSR	0.508	90.5	8/10
BB	1089_USCrime	C-KAN vanilla	0.406	139.0	8/10
BB	1089_USCrime	PyKAN-1 vanilla	0.411	378.0	7/10
BB	1089_USCrime	PyKAN-2 vanilla	0.158	3689.0	3/10
BB	1089_USCrime	PySR	0.712	22.0	10/10
BB	1089_USCrime	Operon	0.113	10.0	10/10
BB	1089_USCrime	TPSR	0.118	47.5	10/10
BB	1199_BNG_echoMonths	C-KAN + GSR	0.433	14.0	10/10
BB	1199_BNG_echoMonths	C-KAN vanilla	0.390	29.0	9/10
BB	1199_BNG_echoMonths	PyKAN-1 vanilla	0.000	—	0/10
BB	1199_BNG_echoMonths	PyKAN-2 vanilla	0.000	—	0/10
BB	1199_BNG_echoMonths	PySR	0.406	3.0	10/10
BB	1199_BNG_echoMonths	Operon	0.448	57.5	10/10
BB	1199_BNG_echoMonths	TPSR	0.334	49.0	10/10
BB	192_vineyard	C-KAN + GSR	0.483	50.0	9/10
BB	192_vineyard	C-KAN vanilla	0.353	61.0	9/10
BB	192_vineyard	PyKAN-1 vanilla	0.058	135.5	10/10
BB	192_vineyard	PyKAN-2 vanilla	0.078	737.0	10/10
BB	192_vineyard	PySR	0.415	16.0	10/10
BB	192_vineyard	Operon	0.530	10.0	10/10
BB	192_vineyard	TPSR	0.335	32.5	10/10
BB	522_pm10	C-KAN + GSR	0.203	61.0	10/10
BB	522_pm10	C-KAN vanilla	0.151	74.0	9/10
BB	522_pm10	PyKAN-1 vanilla	0.000	344.0	10/10
BB	522_pm10	PyKAN-2 vanilla	0.000	2643.5	10/10
BB	522_pm10	PySR	0.105	3.0	10/10
BB	522_pm10	Operon	0.202	52.5	10/10
BB	522_pm10	TPSR	0.017	34.0	10/10
BB	650_fri_c0_500_50	C-KAN + GSR	0.000	—	0/10

Family	Dataset	Method	Accuracy ↑	Size ↓	Valid
BB	650_fri_c0_500_50	C-KAN vanilla	0.000	—	0/10
BB	650_fri_c0_500_50	PyKAN-1 vanilla	0.652	1151.5	10/10
BB	650_fri_c0_500_50	PyKAN-2 vanilla	0.390	10706.0	10/10
BB	650_fri_c0_500_50	PySR	0.780	15.5	10/10
BB	650_fri_c0_500_50	Operon	0.956	34.5	10/10
BB	650_fri_c0_500_50	TPSR	0.266	53.0	10/10
FP	first_principles_absorption	C-KAN + GSR	0.983	30.0	9/10
FP	first_principles_absorption	C-KAN vanilla	0.897	15.0	10/10
FP	first_principles_absorption	PyKAN-1 vanilla	-12.543	112.0	9/10
FP	first_principles_absorption	PyKAN-2 vanilla	-1362.097	1233.0	8/10
FP	first_principles_absorption	PySR	0.978	26.5	10/10
FP	first_principles_absorption	Operon	0.766	8.0	10/10
FP	first_principles_absorption	TPSR	0.956	18.5	10/10
FP	first_principles_ideal_gas	C-KAN + GSR	0.861	41.0	10/10
FP	first_principles_ideal_gas	C-KAN vanilla	0.878	51.5	10/10
FP	first_principles_ideal_gas	PyKAN-1 vanilla	0.699	178.5	10/10
FP	first_principles_ideal_gas	PyKAN-2 vanilla	-0.334	1123.5	10/10
FP	first_principles_ideal_gas	PySR	0.915	29.5	10/10
FP	first_principles_ideal_gas	Operon	0.902	14.0	10/10
FP	first_principles_ideal_gas	TPSR	0.760	25.0	10/10
FP	first_principles_kepler	C-KAN + GSR	0.643	15.0	10/10
FP	first_principles_kepler	C-KAN vanilla	0.198	15.0	10/10
FP	first_principles_kepler	PyKAN-1 vanilla	-22.586	—	0/10
FP	first_principles_kepler	PyKAN-2 vanilla	-22.586	—	0/10
FP	first_principles_kepler	PySR	0.871	22.5	10/10
FP	first_principles_kepler	Operon	0.822	3.0	10/10
FP	first_principles_kepler	TPSR	0.977	16.0	10/10
FP	first_principles_rydberg	C-KAN + GSR	0.991	28.5	10/10
FP	first_principles_rydberg	C-KAN vanilla	0.991	26.0	10/10
FP	first_principles_rydberg	PyKAN-1 vanilla	0.910	115.0	10/10
FP	first_principles_rydberg	PyKAN-2 vanilla	0.766	810.5	10/10
FP	first_principles_rydberg	PySR	0.999	27.0	10/10
FP	first_principles_rydberg	Operon	1.000	19.5	10/10
FP	first_principles_rydberg	TPSR	0.992	30.5	10/10
FP	first_principles_schechter	C-KAN + GSR	0.850	29.5	8/10
FP	first_principles_schechter	C-KAN vanilla	0.837	41.0	9/10
FP	first_principles_schechter	PyKAN-1 vanilla	-2.234	61.5	10/10

Family	Dataset	Method	Accuracy ↑	Size ↓	Valid
FP	first_principles_schechter	PyKAN-2 vanilla	-171.425	343.0	7/10
FP	first_principles_schechter	PySR	0.990	25.0	10/10
FP	first_principles_schechter	Operon	0.969	11.0	10/10
FP	first_principles_schechter	TPSR	0.956	18.0	10/10
FP	first_principles_supernovae_zr	C-KAN + GSR	0.999	93.0	10/10
FP	first_principles_supernovae_zr	C-KAN vanilla	0.998	96.0	10/10
FP	first_principles_supernovae_zr	PyKAN-1 vanilla	-1.818	91.0	10/10
FP	first_principles_supernovae_zr	PyKAN-2 vanilla	0.030	485.0	10/10
FP	first_principles_supernovae_zr	PySR	0.991	29.5	10/10
FP	first_principles_supernovae_zr	Operon	0.999	39.5	10/10
FP	first_principles_supernovae_zr	TPSR	0.813	32.0	10/10

Table 17: Exploratory Track B recovery and bootstrap intervals by difficulty

Method	Easy	Medium	Hard	Weighted overall
C-KAN + GSR	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]
C-KAN vanilla	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]
PyKAN-1 vanilla	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]
PyKAN-2 vanilla	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]
KAN-SR	20.0% [0.0%,50.0%]	20.0% [0.0%,50.0%]	10.0% [0.0%,30.0%]	15.8% [4.2%,30.0%]
PySR	80.0% [50.0%,100.0%]	70.0% [40.0%,100.0%]	0.0% [0.0%,0.0%]	43.3% [31.7%,55.0%]
Operon	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]
TPSR	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]	0.0% [0.0%,0.0%]

Entries are recovery rates [95% stratified bootstrap interval]. The same sampled equation indices are reused across methods; algorithm seeds are selected before resampling.

Table 18: Exploratory Track B run-level recovery and search-potential diagnostics

Method	SR^{run}	SR^{val}	SR^{any}	Fully eval. runs	Median time (s)	Size n
C-KAN + GSR	0.0%	0.0%	0.0%	50.3%	3530	13
C-KAN vanilla	0.0%	0.0%	0.0%	37.0%	3422	13
PyKAN-1 vanilla	0.0%	0.0%	0.0%	37.3%	171	15
PyKAN-2 vanilla	0.0%	0.0%	0.0%	1.3%	401	1
KAN-SR	16.3%	15.8%	18.3%	91.3%	314	29
PySR	42.5%	43.3%	45.8%	96.7%	3057	29
Operon	0.0%	0.0%	0.0%	62.7%	164	23
TPSR	0.0%	0.0%	0.0%	85.0%	271	25

SR^{any} is an oracle search-potential diagnostic, not deployable performance. Timing combines heterogeneous machines and is descriptive only.

P TRACK B RESULTS AND DIAGNOSTICS

The tables in this appendix were generated from the append-only Track B ledger using the frozen 30-equation panel and first five official seeds. The revised comparison includes both C-KAN extraction modes and both standalone vanilla PyKAN topologies. Every displayed method has a complete selected seed block and matching official clean inputs. The PyKAN-vanilla source files configured a 5,000-second fit limit, but all selected terminal runs finished under 3,600 seconds; this nonbinding setting difference keeps the comparison provisional. The changed roster and the tests below are exploratory. Difficulty-weighted overall values use the original SRSD weights $30/120$, $40/120$ and $50/120$.

Table 17 gives the equation-bootstrap intervals behind Figure 4A. With ten equations per group, one equation moves a difficulty-level rate by ten percentage points, so the per-difficulty intervals are wide. Even PySR's Easy interval spans 50–100%.

Table 18 separates three recovery notions. Run-level recovery SR^{run} averages over all five seeds. SR^{val} is the deployable, validation-selected rate. The oracle SR^{any} asks whether any seed recovered the skeleton. For PySR and KAN-SR the three agree to within 3.5 percentage points, so validation selection loses little of their search potential. For C-KAN, Operon and TPSR all three are zero, so the missing recoveries cannot be blamed on seed selection.

The main table already includes C-KAN-Vanilla and both PyKAN-Vanilla topologies. The two C-KAN modes share evolution, whereas the PyKAN topologies are separately trained. Table 19 gives exploratory equation-level tests and Table 20 gives paired effect sizes; all-zero recovery for C-KAN and PyKAN means their McNemar comparison has no discordant equations. These analyses do not repair the different configured PyKAN time limit.

Per-equation outcomes. Table 22 lists every validation-selected outcome. Two features of the scoring are visible in it. First, pointwise validation MSRE (Eq. (31)) is dominated by targets near zero. For example, C-KAN's selection on `feynman-i.6.20a` has test $R^2 = 1.000$ but a validation MSRE of order 10^{34}, because the Gaussian target approaches zero in its tails. Second, NED is capped at one, so a selected equation that is numerically exact but carries a few superfluous terms

Table 19: Exploratory revised dataset-level Track B statistical comparisons

Endpoint	Baseline	Omnibus p	Discordant / W_+	Raw p	Holm p
Skeleton recovery	C-KAN vanilla	1.41×10^{-15}	0/0	1.000	1.000
	PyKAN-1 vanilla		0/0	1.000	1.000
	PyKAN-2 vanilla		0/0	1.000	1.000
	KAN-SR		0/5	0.062	0.375
	PySR		0/15	6.10×10^{-5}	4.27×10^{-4}
	Operon		0/0	1.000	1.000
	TPSR		0/0	1.000	1.000
NED	C-KAN vanilla	1.33×10^{-25}	86.0	0.376	0.752
	PyKAN-1 vanilla		140.0	0.063	0.252
	PyKAN-2 vanilla		140.0	0.063	0.252
	KAN-SR		15.0	10.00×10^{-6}	7.00×10^{-5}
	PySR		8.0	10.00×10^{-6}	7.00×10^{-5}
	Operon		70.5	0.007	0.035
	TPSR		110.0	0.625	0.752

Tests use equations as paired units. McNemar tests are exact and two-sided; NED uses Pratt-ranked two-sided sign-flip permutation p values. Holm correction is separate within each family; pairwise claims require the corresponding omnibus test.

Table 20: Exploratory C-KAN paired effect sizes against Track B baselines

Baseline	ΔSR (pp)	95% CI (pp)	Median ΔNED	95% CI	Rank-biserial
C-KAN vanilla	+0.0	[+0.0,+0.0]	+0.000	[+0.000,+0.000]	+0.509
PyKAN-1 vanilla	+0.0	[+0.0,+0.0]	+0.000	[+0.000,+0.000]	+1.000
PyKAN-2 vanilla	+0.0	[+0.0,+0.0]	+0.000	[+0.000,+0.000]	+1.000
KAN-SR	-15.8	[-30.0,-4.2]	-0.362	[-0.667,-0.174]	-0.933
PySR	-43.3	[-55.0,-31.7]	-0.708	[-1.000,-0.296]	-0.965
Operon	+0.0	[+0.0,+0.0]	+0.000	[-0.247,+0.000]	-0.636
TPSR	+0.0	[+0.0,+0.0]	+0.000	[+0.000,+0.000]	+0.164

Positive differences favor C-KAN. Recovery differences use difficulty weights; median NED differences and rank-biserial effects use observed paired equations. Bootstrap intervals resample equations within difficulty with fixed counts.

can score the same as an unrelated equation. Both are properties of the prespecified protocol, applied equally to all methods. For readability, fixed-point values with six or more integer digits are shown in three-significant-digit scientific notation.

Table 21: First stage at which C-KAN-GSR loses each clean Track B equation (best of five seeds). Category D uses the post-hoc clean-up and is a diagnostic, not a benchmark result.

First stage at which the equation is lost	Easy	Medium	Hard	Total
A. Search: no accurate prepared network	4	7	6	17
B. Conversion: accurate network, no accurate symbolic candidate	1	1	1	3
C. Export/structure: accurate equation, not recovered even after clean-up	3	1	3	7
D. Recovered only after post-hoc clean-up	2	1	0	3

Table 22: Exploratory Track B validation-selected outcome for each SRSD equation

Difficulty	Equation	Method	Seed	SR^{val}	NED	Test R^2	Val. MSRE	Size ratio
Easy	feynman-i.12.1	C-KAN + GSR	23654	0	1.000	1.000	6.90×10^{-6}	84.00
Easy	feynman-i.12.1	C-KAN vanilla	5390	0	1.000	0.998	2.16108	—
Easy	feynman-i.12.1	PyKAN-1 vanilla	16850	0	1.000	0.958	104.71799	57.33
Easy	feynman-i.12.1	PyKAN-2 vanilla	5390	0	1.000	0.385	1626.28597	—
Easy	feynman-i.12.1	KAN-SR	23654	0	0.333	1.000	2.02×10^{-31}	1.33
Easy	feynman-i.12.1	PySR	23654	1	0.000	1.000	0	1.00
Easy	feynman-i.12.1	Operon	16850	0	1.000	1.000	1.04×10^{-12}	2.00
Easy	feynman-i.12.1	TPSR	23654	0	1.000	0.813	0.09626	3.00
Easy	feynman-i.14.4	C-KAN + GSR	16850	0	1.000	1.000	0.00099	72.17
Easy	feynman-i.14.4	C-KAN vanilla	16850	0	1.000	0.999	7289.99598	31.83
Easy	feynman-i.14.4	PyKAN-1 vanilla	860	0	1.000	0.858	111393.41431	69.50
Easy	feynman-i.14.4	PyKAN-2 vanilla	15795	0	1.000	0.457	3843619.74507	—
Easy	feynman-i.14.4	KAN-SR	23654	1	0.000	1.000	1.24×10^{-30}	1.00
Easy	feynman-i.14.4	PySR	23654	1	0.000	1.000	0	1.00
Easy	feynman-i.14.4	Operon	16850	0	0.333	1.000	1.76×10^{-15}	1.33
Easy	feynman-i.14.4	TPSR	16850	0	1.000	0.376	1195578.44300	81.50
Easy	feynman-i.18.16	C-KAN + GSR	16850	0	1.000	0.933	6053.76607	—
Easy	feynman-i.18.16	C-KAN vanilla	23654	0	1.000	0.454	2476372.82321	—
Easy	feynman-i.18.16	PyKAN-1 vanilla	5390	0	1.000	0.125	471444.78906	—
Easy	feynman-i.18.16	PyKAN-2 vanilla	860	0	1.000	-5.678	3049167643.02061	—
Easy	feynman-i.18.16	KAN-SR	23654	0	0.500	1.000	2.17×10^{-26}	1.50
Easy	feynman-i.18.16	PySR	23654	1	0.000	1.000	9.49×10^{-34}	1.00
Easy	feynman-i.18.16	Operon	5390	0	0.500	1.000	7.97×10^{-16}	1.50
Easy	feynman-i.18.16	TPSR	—	0	1.000	—	—	—
Easy	feynman-i.25.13	C-KAN + GSR	15795	0	1.000	0.992	1.10228	15.00
Easy	feynman-i.25.13	C-KAN vanilla	5390	0	1.000	0.970	4.06835	—
Easy	feynman-i.25.13	PyKAN-1 vanilla	5390	0	1.000	0.490	774.48148	—
Easy	feynman-i.25.13	PyKAN-2 vanilla	16850	0	1.000	0.450	4822.32583	—
Easy	feynman-i.25.13	KAN-SR	860	0	0.200	1.000	3.17×10^{-30}	1.20
Easy	feynman-i.25.13	PySR	23654	1	0.000	1.000	0	1.00
Easy	feynman-i.25.13	Operon	16850	0	0.400	1.000	1.05×10^{-13}	1.40
Easy	feynman-i.25.13	TPSR	23654	0	1.000	-0.000	1.01248	0.60
Easy	feynman-i.27.6	C-KAN + GSR	16850	0	1.000	0.944	1.02657	—
Easy	feynman-i.27.6	C-KAN vanilla	860	0	1.000	0.939	2.69705	—
Easy	feynman-i.27.6	PyKAN-1 vanilla	23654	0	1.000	0.745	13.18529	—
Easy	feynman-i.27.6	PyKAN-2 vanilla	860	0	1.000	0.555	57.65623	—
Easy	feynman-i.27.6	KAN-SR	23654	0	0.200	1.000	5.59×10^{-31}	1.20
Easy	feynman-i.27.6	PySR	15795	0	1.000	1.000	3.94×10^{-33}	1.70
Easy	feynman-i.27.6	Operon	860	0	1.000	0.999	0.04252	10.00
Easy	feynman-i.27.6	TPSR	23654	0	1.000	0.315	30.05340	24.60
Easy	feynman-i.47.23	C-KAN + GSR	23654	0	1.000	1.000	3.76×10^{-6}	—
Easy	feynman-i.47.23	C-KAN vanilla	23654	0	1.000	1.000	7.94×10^{-6}	—
Easy	feynman-i.47.23	PyKAN-1 vanilla	5390	0	1.000	0.989	0.00040	14.50
Easy	feynman-i.47.23	PyKAN-2 vanilla	16850	0	1.000	0.905	0.00545	—
Easy	feynman-i.47.23	KAN-SR	23654	0	1.000	1.000	2.83×10^{-6}	2.00
Easy	feynman-i.47.23	PySR	23654	1	0.000	1.000	0	1.00
Easy	feynman-i.47.23	Operon	23654	0	0.500	1.000	1.91×10^{-15}	1.50
Easy	feynman-i.47.23	TPSR	—	0	1.000	—	—	—
Easy	feynman-ii.2.42	C-KAN + GSR	16850	0	1.000	0.317	16423439.38352	—
Easy	feynman-ii.2.42	C-KAN vanilla	860	0	1.000	0.199	806652344.42062	—

Difficulty	Equation	Method	Seed	SR^{val}	NED	Test R^2	Val. MSRE	Size ratio
Easy	feynman-ii.2.42	PyKAN-1 vanilla	23654	0	1.000	0.813	1256042769.18023	—
Easy	feynman-ii.2.42	PyKAN-2 vanilla	860	0	1.000	0.114	1355671153.52808	—
Easy	feynman-ii.2.42	KAN-SR	15795	0	0.091	1.000	7.00×10^{-31}	1.09
Easy	feynman-ii.2.42	PySR	23654	1	0.000	1.000	9.51×10^{-33}	1.00
Easy	feynman-ii.2.42	Operon	860	0	1.000	1.000	4.45×10^{-15}	1.64
Easy	feynman-ii.2.42	TPSR	23654	0	1.000	0.003	40938695.30663	2.18
Easy	feynman-ii.8.31	C-KAN + GSR	16850	0	1.000	1.000	1.15×10^{-10}	2.00
Easy	feynman-ii.8.31	C-KAN vanilla	860	0	1.000	1.000	0.00203	5.20
Easy	feynman-ii.8.31	PyKAN-1 vanilla	16850	0	1.000	0.979	3805.76192	14.60
Easy	feynman-ii.8.31	PyKAN-2 vanilla	15795	0	1.000	-0.156	98356.06918	—
Easy	feynman-ii.8.31	KAN-SR	23654	1	0.000	1.000	2.69×10^{-30}	1.00
Easy	feynman-ii.8.31	PySR	23654	1	0.000	1.000	0	1.00
Easy	feynman-ii.8.31	Operon	15795	0	0.400	1.000	1.64×10^{-10}	1.40
Easy	feynman-ii.8.31	TPSR	—	0	1.000	—	—	—
Easy	feynman-ii.15.4	C-KAN + GSR	16850	0	1.000	1.000	3.03×10^{-5}	—
Easy	feynman-ii.15.4	C-KAN vanilla	860	0	1.000	0.037	528.81753	42.00
Easy	feynman-ii.15.4	PyKAN-1 vanilla	16850	0	1.000	-0.786	16797.51814	—
Easy	feynman-ii.15.4	PyKAN-2 vanilla	—	0	1.000	—	—	—
Easy	feynman-ii.15.4	KAN-SR	15795	0	0.833	1.000	3.60×10^{-27}	1.67
Easy	feynman-ii.15.4	PySR	23654	0	0.667	1.000	8.90×10^{-34}	1.50
Easy	feynman-ii.15.4	Operon	—	0	1.000	—	—	—
Easy	feynman-ii.15.4	TPSR	23654	0	1.000	-3034697388893835584219068944324807884800.000	28706841809745448923862268153186646107357184.00000	8.50
Easy	feynman-ii.38.14	C-KAN + GSR	23654	0	1.000	1.000	0.00433	11.88
Easy	feynman-ii.38.14	C-KAN vanilla	23654	0	1.000	1.000	0.00049	—
Easy	feynman-ii.38.14	PyKAN-1 vanilla	23654	0	1.000	0.999	0.08769	19.62
Easy	feynman-ii.38.14	PyKAN-2 vanilla	5390	0	1.000	0.182	23.11250	—
Easy	feynman-ii.38.14	KAN-SR	15795	0	0.250	1.000	3.64×10^{-31}	1.25
Easy	feynman-ii.38.14	PySR	23654	1	0.000	1.000	0	1.00
Easy	feynman-ii.38.14	Operon	5390	0	1.000	1.000	3.58×10^{-17}	1.75
Easy	feynman-ii.38.14	TPSR	23654	0	1.000	0.983	0.00870	1.75
Medium	feynman-i.8.14	C-KAN + GSR	5390	0	1.000	0.984	0.35898	—
Medium	feynman-i.8.14	C-KAN vanilla	16850	0	1.000	0.966	0.60207	2.30
Medium	feynman-i.8.14	PyKAN-1 vanilla	16850	0	1.000	0.674	3.21087	—
Medium	feynman-i.8.14	PyKAN-2 vanilla	16850	0	1.000	-0.617	0.58001	—
Medium	feynman-i.8.14	KAN-SR	16850	0	0.870	0.958	0.87878	1.04
Medium	feynman-i.8.14	PySR	23654	1	0.000	1.000	0	1.00
Medium	feynman-i.8.14	Operon	16850	0	1.000	0.991	0.18073	5.57
Medium	feynman-i.8.14	TPSR	23654	0	0.870	0.401	0.26039	1.09
Medium	feynman-i.11.19	C-KAN + GSR	16850	0	1.000	0.415	10.54107	36.40
Medium	feynman-i.11.19	C-KAN vanilla	16850	0	1.000	0.397	24.46449	87.00
Medium	feynman-i.11.19	PyKAN-1 vanilla	16850	0	1.000	0.990	8.85548	21.00
Medium	feynman-i.11.19	PyKAN-2 vanilla	5390	0	1.000	0.892	75.89415	—
Medium	feynman-i.11.19	KAN-SR	23654	1	0.000	1.000	1.17×10^{-29}	1.00
Medium	feynman-i.11.19	PySR	23654	1	0.000	1.000	0	1.00
Medium	feynman-i.11.19	Operon	5390	0	0.900	1.000	4.08×10^{-15}	1.40
Medium	feynman-i.11.19	TPSR	16850	0	1.000	-0.001	2.84876	3.20
Medium	feynman-i.12.11	C-KAN + GSR	860	0	1.000	0.873	12.40789	—
Medium	feynman-i.12.11	C-KAN vanilla	5390	0	1.000	0.731	210.81637	8.44
Medium	feynman-i.12.11	PyKAN-1 vanilla	15795	0	1.000	-0.603	3764.07977	—
Medium	feynman-i.12.11	PyKAN-2 vanilla	23654	0	1.000	0.329	1493.87018	—
Medium	feynman-i.12.11	KAN-SR	23654	0	0.222	1.000	5.75×10^{-31}	1.22
Medium	feynman-i.12.11	PySR	23654	1	0.000	1.000	6.85×10^{-34}	1.00

Difficulty	Equation	Method	Seed	SR^{val}	NED	Test R^2	Val. MSRE	Size ratio
Medium	feynman-i.12.11	Operon	860	0	0.778	1.000	2.09×10^{-13}	1.44
Medium	feynman-i.12.11	TPSR	23654	0	1.000	0.770	1.04420	9.11
Medium	feynman-i.16.6	C-KAN + GSR	860	0	0.583	0.999	0.00062	0.67
Medium	feynman-i.16.6	C-KAN vanilla	15795	0	1.000	1.000	0.00017	1.58
Medium	feynman-i.16.6	PyKAN-1 vanilla	16850	0	1.000	0.977	6.52997	11.50
Medium	feynman-i.16.6	PyKAN-2 vanilla	23654	0	1.000	0.941	237.28744	—
Medium	feynman-i.16.6	KAN-SR	23654	0	0.667	0.999	0.00028	0.58
Medium	feynman-i.16.6	PySR	15795	0	0.750	0.999	0.00013	0.25
Medium	feynman-i.16.6	Operon	15795	0	1.000	1.000	1.96×10^{-7}	1.67
Medium	feynman-i.16.6	TPSR	23654	0	1.000	-92433857782920.094	416677543358567.43750	4.42
Medium	feynman-i.18.4	C-KAN + GSR	23654	0	1.000	0.982	17.09513	—
Medium	feynman-i.18.4	C-KAN vanilla	5390	0	1.000	0.897	322.89130	—
Medium	feynman-i.18.4	PyKAN-1 vanilla	23654	0	1.000	0.969	34.31446	—
Medium	feynman-i.18.4	PyKAN-2 vanilla	15795	0	1.000	0.892	673.87935	—
Medium	feynman-i.18.4	KAN-SR	16850	0	1.000	0.891	74.67981	6.15
Medium	feynman-i.18.4	PySR	23654	1	0.000	1.000	1.91×10^{-30}	1.00
Medium	feynman-i.18.4	Operon	5390	0	1.000	—	0.08165	—
Medium	feynman-i.18.4	TPSR	23654	0	1.000	0.665	595.74601	2.23
Medium	feynman-i.29.4	C-KAN + GSR	5390	0	0.667	1.000	6.90×10^{-15}	1.67
Medium	feynman-i.29.4	C-KAN vanilla	16850	0	0.667	1.000	1.77×10^{-13}	1.67
Medium	feynman-i.29.4	PyKAN-1 vanilla	860	0	1.000	0.997	0.37283	40.67
Medium	feynman-i.29.4	PyKAN-2 vanilla	23654	0	1.000	0.867	0.07972	—
Medium	feynman-i.29.4	KAN-SR	23654	1	0.000	1.000	5.56×10^{-32}	1.00
Medium	feynman-i.29.4	PySR	23654	1	0.000	1.000	3.36×10^{-33}	1.00
Medium	feynman-i.29.4	Operon	5390	0	0.667	1.000	2.96×10^{-13}	1.67
Medium	feynman-i.29.4	TPSR	860	0	1.000	-2525152586276244480.000	183129408143944581120.00000	12.33
Medium	feynman-ii.6.11	C-KAN + GSR	16850	0	1.000	0.173	16821.14152	—
Medium	feynman-ii.6.11	C-KAN vanilla	860	0	1.000	-0.000	309242.11577	—
Medium	feynman-ii.6.11	PyKAN-1 vanilla	16850	0	1.000	-55.246	1777202079.02196	35.75
Medium	feynman-ii.6.11	PyKAN-2 vanilla	—	0	1.000	—	—	—
Medium	feynman-ii.6.11	KAN-SR	5390	0	0.250	1.000	1.79×10^{-26}	1.25
Medium	feynman-ii.6.11	PySR	23654	1	0.000	1.000	1.10×10^{-33}	1.00
Medium	feynman-ii.6.11	Operon	23654	0	0.500	1.000	1.39×10^{-5}	1.50
Medium	feynman-ii.6.11	TPSR	23654	0	0.750	-0.000	1.00000	0.38
Medium	feynman-iii.4.32	C-KAN + GSR	860	0	1.000	0.994	10.72803	—
Medium	feynman-iii.4.32	C-KAN vanilla	860	0	1.000	0.993	23.11735	—
Medium	feynman-iii.4.32	PyKAN-1 vanilla	23654	0	1.000	0.566	1523.44005	53.09
Medium	feynman-iii.4.32	PyKAN-2 vanilla	16850	0	1.000	-0.238	6512.72217	—
Medium	feynman-iii.4.32	KAN-SR	23654	0	0.636	1.000	1.60×10^{-7}	0.55
Medium	feynman-iii.4.32	PySR	23654	0	1.000	1.000	6.25×10^{-21}	2.45
Medium	feynman-iii.4.32	Operon	15795	0	0.455	1.000	3.38×10^{-11}	0.73
Medium	feynman-iii.4.32	TPSR	15795	0	1.000	-193285622.030	8819.08946	8.73
Medium	feynman-iii.15.12	C-KAN + GSR	—	0	1.000	—	—	—
Medium	feynman-iii.15.12	C-KAN vanilla	—	0	1.000	—	—	—
Medium	feynman-iii.15.12	PyKAN-1 vanilla	—	0	1.000	—	—	—
Medium	feynman-iii.15.12	PyKAN-2 vanilla	—	0	1.000	—	—	—
Medium	feynman-iii.15.12	KAN-SR	—	0	1.000	—	—	—
Medium	feynman-iii.15.12	PySR	—	0	1.000	—	—	—
Medium	feynman-iii.15.12	Operon	—	0	1.000	—	—	—
Medium	feynman-iii.15.12	TPSR	—	0	1.000	—	—	—
Medium	feynman-bonus.18	C-KAN + GSR	16850	0	1.000	0.999	755.48963	—
Medium	feynman-bonus.18	C-KAN vanilla	23654	0	1.000	0.992	15329.69011	—

Difficulty	Equation	Method	Seed	SR^{val}	NED	Test R^2	Val. MSRE	Size ratio
Medium	feynman-bonus.18	PyKAN-1 vanilla	16850	0	1.000	0.910	109765.34436	—
Medium	feynman-bonus.18	PyKAN-2 vanilla	5390	0	1.000	0.727	7930541.08419	—
Medium	feynman-bonus.18	KAN-SR	860	0	1.000	0.999	1043.42987	1.92
Medium	feynman-bonus.18	PySR	15795	1	0.000	1.000	4.34×10^{-30}	1.00
Medium	feynman-bonus.18	Operon	16850	0	1.000	0.931	111759.49332	3.58
Medium	feynman-bonus.18	TPSR	23654	0	1.000	0.305	7851.20800	3.25
Hard	feynman-i.6.20a	C-KAN + GSR	860	0	1.000	1.000	14394364689041117894707766477455360.00000	7.50
Hard	feynman-i.6.20a	C-KAN vanilla	860	0	1.000	1.000	631045847794697410764532416323256 32.00000	3.50
Hard	feynman-i.6.20a	PyKAN-1 vanilla	16850	0	1.000	0.105	873861778912019703200312367917669181030 4.00000	8.00
Hard	feynman-i.6.20a	PyKAN-2 vanilla	5390	0	1.000	-0.592	37388903598502927165958141103671556964352.00000	—
Hard	feynman-i.6.20a	KAN-SR	23654	1	0.000	1.000	5.92×10^{-29}	1.00
Hard	feynman-i.6.20a	PySR	23654	0	0.250	1.000	8.87×10^{-33}	1.25
Hard	feynman-i.6.20a	Operon	16850	0	0.875	1.000	40273800406262706037850112.00000	1.50
Hard	feynman-i.6.20a	TPSR	23654	0	1.000	0.938	24152456980007900921297832721262418 3296.00000	2.75
Hard	feynman-i.30.3	C-KAN + GSR	15795	0	0.833	0.237	6578033327048.24023	1.06
Hard	feynman-i.30.3	C-KAN vanilla	15795	0	0.722	-0.195	9943471146994.72266	0.56
Hard	feynman-i.30.3	PyKAN-1 vanilla	15795	0	1.000	-3.085	9064412214288792.00000	—
Hard	feynman-i.30.3	PyKAN-2 vanilla	15795	0	1.000	-3.062	320745182405606.12500	—
Hard	feynman-i.30.3	KAN-SR	15795	0	0.611	-0.028	241264725206.52527	1.11
Hard	feynman-i.30.3	PySR	860	0	0.500	-0.002	108.97693	0.56
Hard	feynman-i.30.3	Operon	—	0	1.000	—	—	—
Hard	feynman-i.30.3	TPSR	23654	0	0.778	-0.001	510362926958.32635	0.67
Hard	feynman-i.44.4	C-KAN + GSR	23654	0	1.000	0.917	490.91960	—
Hard	feynman-i.44.4	C-KAN vanilla	—	0	1.000	—	—	—
Hard	feynman-i.44.4	PyKAN-1 vanilla	16850	0	1.000	-0.495	24144.60997	—
Hard	feynman-i.44.4	PyKAN-2 vanilla	23654	0	1.000	-1.497	1876370.95037	—
Hard	feynman-i.44.4	KAN-SR	5390	0	1.000	1.000	5.10×10^{-30}	1.30
Hard	feynman-i.44.4	PySR	860	0	0.900	0.729	0.47741	1.10
Hard	feynman-i.44.4	Operon	16850	0	0.400	1.000	7.02×10^{-13}	1.20
Hard	feynman-i.44.4	TPSR	—	0	1.000	—	—	—
Hard	feynman-ii.13.23	C-KAN + GSR	23654	0	1.000	1.000	4.36×10^{-5}	—
Hard	feynman-ii.13.23	C-KAN vanilla	15795	0	1.000	1.000	3.57×10^{-5}	2.82
Hard	feynman-ii.13.23	PyKAN-1 vanilla	5390	0	1.000	0.999	0.00305	6.27
Hard	feynman-ii.13.23	PyKAN-2 vanilla	5390	0	1.000	0.841	1.05070	44.64
Hard	feynman-ii.13.23	KAN-SR	23654	0	0.818	1.000	0.00016	0.27
Hard	feynman-ii.13.23	PySR	23654	0	0.727	1.000	3.29×10^{-11}	1.36
Hard	feynman-ii.13.23	Operon	15795	0	0.636	0.993	0.25386	0.45
Hard	feynman-ii.13.23	TPSR	23654	0	1.000	-45.259	27.47567	3.82
Hard	feynman-ii.35.21	C-KAN + GSR	15795	0	1.000	0.948	85071312.57647	—
Hard	feynman-ii.35.21	C-KAN vanilla	15795	0	1.000	0.780	255776203.91639	—
Hard	feynman-ii.35.21	PyKAN-1 vanilla	23654	0	1.000	—	7393826886.17955	—
Hard	feynman-ii.35.21	PyKAN-2 vanilla	16850	0	1.000	—	4280303713.71090	—
Hard	feynman-ii.35.21	KAN-SR	23654	0	0.636	1.000	1.39×10^{-11}	0.91
Hard	feynman-ii.35.21	PySR	860	0	0.909	0.852	0.76905	1.36
Hard	feynman-ii.35.21	Operon	860	0	0.727	1.000	0.02137	1.09
Hard	feynman-ii.35.21	TPSR	23654	0	1.000	-0.000	8313872309.75686	4.27
Hard	feynman-iii.9.52	C-KAN + GSR	5390	0	1.000	0.999	101268160112.74776	—
Hard	feynman-iii.9.52	C-KAN vanilla	23654	0	1.000	0.991	5643701122.98434	—
Hard	feynman-iii.9.52	PyKAN-1 vanilla	860	0	1.000	0.192	33736924753227.74219	—
Hard	feynman-iii.9.52	PyKAN-2 vanilla	15795	0	1.000	0.516	107605021976427.54688	—
Hard	feynman-iii.9.52	KAN-SR	23654	0	0.640	1.000	3.63×10^{-29}	0.44
Hard	feynman-iii.9.52	PySR	16850	0	0.800	0.244	1567576.73169	0.24
Hard	feynman-iii.9.52	Operon	—	0	1.000	—	—	—

Difficulty	Equation	Method	Seed	SR^{val}	NED	Test R^2	Val. MSRE	Size ratio
Hard	feynman-iii.9.52	TPSR	860	0	1.000	-0.012	87425765578545.12500	1.16
Hard	feynman-bonus.6	C-KAN + GSR	23654	0	0.875	0.015	113425765.88422	0.46
Hard	feynman-bonus.6	C-KAN vanilla	5390	0	1.000	0.049	308151437.95153	3.29
Hard	feynman-bonus.6	PyKAN-1 vanilla	860	0	1.000	-1692.818	746205184.42389	30.33
Hard	feynman-bonus.6	PyKAN-2 vanilla	5390	0	1.000	-0.971	3333344604.09752	—
Hard	feynman-bonus.6	KAN-SR	16850	0	1.000	0.500	111610904.53410	5.12
Hard	feynman-bonus.6	PySR	15795	0	0.750	0.011	1.05305	0.62
Hard	feynman-bonus.6	Operon	16850	0	1.000	0.533	429347920.19843	1.50
Hard	feynman-bonus.6	TPSR	23654	0	1.000	-0.002	475265259.80788	3.58
Hard	feynman-bonus.12	C-KAN + GSR	23654	0	0.865	-0.001	25496886808265848.00000	0.21
Hard	feynman-bonus.12	C-KAN vanilla	15795	0	0.923	-0.001	863103058110293.75000	0.19
Hard	feynman-bonus.12	PyKAN-1 vanilla	—	0	1.000	—	—	—
Hard	feynman-bonus.12	PyKAN-2 vanilla	23654	0	1.000	-0.372	10153880302457664307 2.00000	—
Hard	feynman-bonus.12	KAN-SR	23654	0	0.846	0.001	207145706881024 80.00000	0.15
Hard	feynman-bonus.12	PySR	5390	0	0.865	0.933	1.01324	0.13
Hard	feynman-bonus.12	Operon	860	0	1.000	—	54102787547.76867	—
Hard	feynman-bonus.12	TPSR	23654	0	0.923	-0.001	3985458723448149.00000	0.10
Hard	feynman-bonus.17	C-KAN + GSR	23654	0	1.000	0.986	35771.33600	14.48
Hard	feynman-bonus.17	C-KAN vanilla	23654	0	1.000	0.983	9185.37591	—
Hard	feynman-bonus.17	PyKAN-1 vanilla	16850	0	1.000	0.373	2643083.79755	6.29
Hard	feynman-bonus.17	PyKAN-2 vanilla	860	0	1.000	0.568	727677.57014	—
Hard	feynman-bonus.17	KAN-SR	23654	0	0.742	1.000	4.48×10^{-31}	0.26
Hard	feynman-bonus.17	PySR	23654	0	0.742	1.000	6.35×10^{-33}	0.26
Hard	feynman-bonus.17	Operon	5390	0	0.839	0.001	2703506.66521	0.16
Hard	feynman-bonus.17	TPSR	23654	0	1.000	-0.099	0.97575	3.32
Hard	feynman-bonus.20	C-KAN + GSR	15795	0	1.000	0.990	3310742.42960	—
Hard	feynman-bonus.20	C-KAN vanilla	23654	0	1.000	0.995	73542445.18550	—
Hard	feynman-bonus.20	PyKAN-1 vanilla	16850	0	1.000	-3.231	9014792960672.92773	13.81
Hard	feynman-bonus.20	PyKAN-2 vanilla	23654	0	1.000	—	238128729.11971	—
Hard	feynman-bonus.20	KAN-SR	16850	0	0.762	1.000	2036.51605	0.90
Hard	feynman-bonus.20	PySR	5390	0	0.571	1.000	0.38260	1.19
Hard	feynman-bonus.20	Operon	—	0	1.000	—	—	—
Hard	feynman-bonus.20	TPSR	23654	0	1.000	-525849553811550839721031271943392937904247290490980925 44.000	1052262032505014637821989945632233769385703577189906731472508682 24.00000	1.81

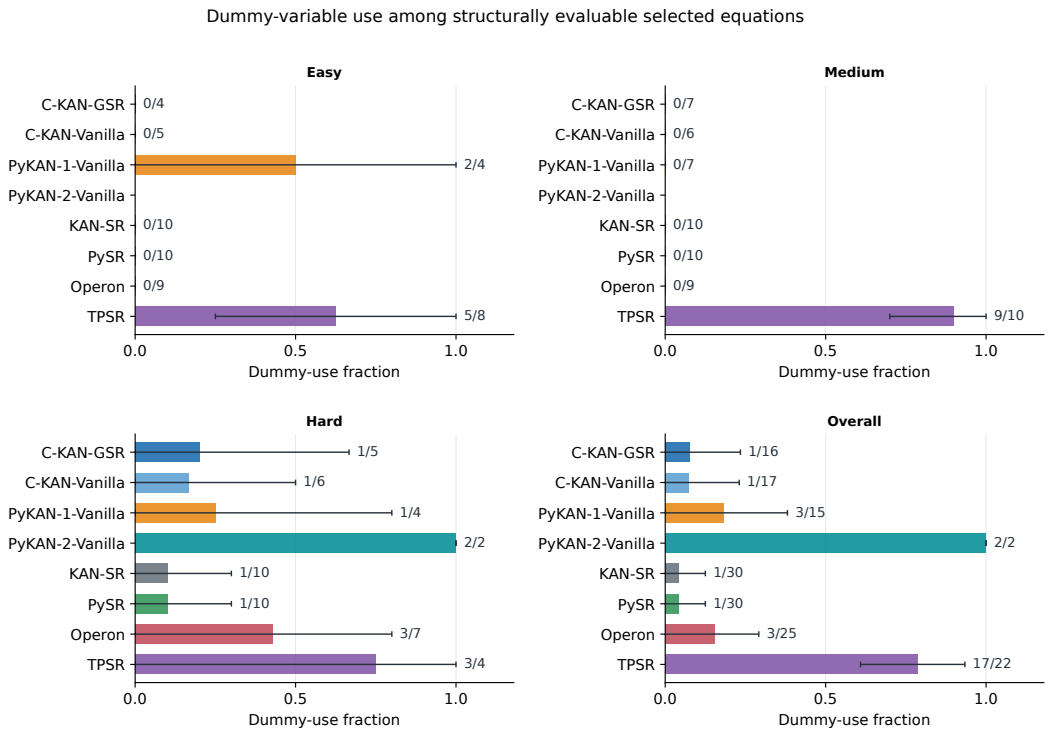

Figure 6: Provisional dummy-variable use among structurally evaluable validation-selected expressions, separated by difficulty. Numerators and denominators are the observed equation counts. The overall bar uses difficulty-weighted conditional counts. A method with no evaluable expression has no diagnostic value for that equation.

Q TRACK C PROVISIONAL RESULTS AND DIAGNOSTICS

The tables and figures in this appendix were generated from the append-only Track C and Track B result ledgers using the frozen short-panel equation and seed lists. They remain provisional because cross-method corrupted-array hashes disagree for all 30 Track C equations. Both vanilla PyKAN topologies have complete seed blocks; PyKAN-GSR remains incomplete and is excluded. Each listed C-KAN extraction mode shares its evolution with the other mode; the rows must not be counted as independent training runs. Difficulty-weighted overall summaries use weights $30/120$, $40/120$ and $50/120$ for Easy, Medium and Hard, whereas the displayed raw numerators and denominators refer to the 30 observed equations.

Selected-variable overlap with Track B. After mapping each clean selected-variable set through Track C's recorded dummy-column insertion indices, the mean B–C Jaccard similarity among pairs evaluable in both tracks is 0.966 for PySR ($n = 29$), 0.911 for KAN-SR ($n = 29$), 0.901 for Operon ($n = 23$), and 0.535 for TPSR ($n = 18$). C-KAN-GSR has 0.631 ($n = 10$), C-KAN-Vanilla 0.789 ($n = 11$), PyKAN-1-Vanilla 0.871 ($n = 11$), and PyKAN-2-Vanilla 0.500 ($n = 1$). Within these conditional subsets, C-KAN has lower observed overlap than PySR, KAN-SR or Operon. The small C-KAN and PyKAN denominators, especially the single PyKAN-2 pair, and the corrupted-input disagreement preclude claims of comparative structural stability or a causal effect of corruption. Variable-set overlap is a supplementary diagnostic, not a substitute for skeleton recovery.

Table 23: Provisional Track C recovery by difficulty, with paired clean Track B values

Method	Difficulty	SR_B^{val}	SR_C^{val}	Mean NED$_B$	Mean NED$_C$	Dummy use$_C$
C-KAN-GSR	Easy	0.0%	0.0%	1.000	1.000	0.0% (0/4)
	Medium	0.0%	0.0%	0.925	0.967	0.0% (0/7)
	Hard	0.0%	0.0%	0.957	0.945	20.0% (1/5)
	Overall	0.0%	0.0%	0.957	0.966	7.7% (1/16)
C-KAN-Vanilla	Easy	0.0%	0.0%	1.000	1.000	0.0% (0/5)
	Medium	0.0%	0.0%	0.967	1.000	0.0% (0/6)
	Hard	0.0%	0.0%	0.965	0.977	16.7% (1/6)
	Overall	0.0%	0.0%	0.974	0.991	7.2% (1/17)
PyKAN-1-Vanilla	Easy	0.0%	0.0%	1.000	1.000	50.0% (2/4)
	Medium	0.0%	0.0%	1.000	1.000	0.0% (0/7)
	Hard	0.0%	0.0%	1.000	1.000	25.0% (1/4)
	Overall	0.0%	0.0%	1.000	1.000	18.3% (3/15)
PyKAN-2-Vanilla	Easy	0.0%	0.0%	1.000	1.000	—
	Medium	0.0%	0.0%	1.000	1.000	—
	Hard	0.0%	0.0%	1.000	1.000	100.0% (2/2)
	Overall	0.0%	0.0%	1.000	1.000	100.0% (2/2)
KAN-SR	Easy	20.0%	20.0%	0.341	0.537	0.0% (0/10)
	Medium	20.0%	10.0%	0.564	0.812	0.0% (0/10)
	Hard	10.0%	0.0%	0.706	0.803	10.0% (1/10)
	Overall	15.8%	8.3%	0.567	0.739	4.2% (1/30)
PySR	Easy	80.0%	80.0%	0.167	0.111	0.0% (0/10)
	Medium	70.0%	60.0%	0.275	0.330	0.0% (0/10)
	Hard	0.0%	0.0%	0.702	0.646	10.0% (1/10)
	Overall	43.3%	40.0%	0.426	0.407	4.2% (1/30)
Operon	Easy	0.0%	0.0%	0.713	0.817	0.0% (0/9)
	Medium	0.0%	0.0%	0.830	0.785	0.0% (0/9)
	Hard	0.0%	0.0%	0.848	0.815	42.9% (3/7)
	Overall	0.0%	0.0%	0.808	0.806	15.3% (3/25)
TPSR	Easy	0.0%	0.0%	1.000	0.982	62.5% (5/8)
	Medium	0.0%	0.0%	0.962	0.973	90.0% (9/10)
	Hard	0.0%	0.0%	0.970	0.960	75.0% (3/4)
	Overall	0.0%	0.0%	0.975	0.970	78.6% (17/22)

Table 24: Provisional Track C run-level and expression diagnostics (supplement)

Method	SR_C^{run}	SR_C^{any}	Valid runs	Size ratio	High-accuracy dummy use	Elapsed (s)
C-KAN-GSR	0.0%	0.0%	47.7%	2.77 [1.67,4.25]	18.5% (1/7)	2623
C-KAN-Vanilla	0.0%	0.0%	43.3%	3.12 [1.68,4.33]	0.0% (0/5)	2583
PyKAN-1-Vanilla	0.0%	0.0%	40.0%	10.00 [6.32,20.18]	—	201
PyKAN-2-Vanilla	0.0%	0.0%	1.7%	9.38 [9.38,17.00]	—	406
KAN-SR	9.3%	10.8%	92.0%	1.18 [0.79,2.12]	0.0% (0/18)	1658
PySR	42.0%	45.8%	94.7%	1.00 [0.54,1.00]	0.0% (0/23)	3041
Operon	0.0%	0.0%	58.0%	1.60 [1.25,1.88]	0.0% (0/18)	146
TPSR	0.0%	0.0%	75.3%	3.50 [0.60,6.62]	—	409

SR^{any} is an oracle diagnostic, not deployable performance. Size ratio is median [Q1,Q3] over structurally evaluable selected equations. Elapsed time is descriptive across mixed machines and should not be used for a controlled runtime comparison.

Table 25: PROVISIONAL Track C selected-variable diagnostics, conditional on structural evaluability.

Method	Evaluable	Precision	Recall	F_1	True kept	True missed	Dummy kept
C-KAN-GSR	16/30	0.969	0.812	0.851	37	14	1
C-KAN-Vanilla	17/30	0.993	0.790	0.849	41	17	1
PyKAN-1-Vanilla	15/30	0.912	0.900	0.872	34	6	5
PyKAN-2-Vanilla	2/30	0.417	1.000	0.583	3	0	4
KAN-SR	30/30	0.983	0.857	0.896	79	21	2
PySR	30/30	0.989	0.897	0.924	84	16	1
Operon	25/30	0.943	0.886	0.901	71	12	3
TPSR	22/30	0.603	0.649	0.574	42	24	26

Precision, recall and F_1 are unweighted means over structurally evaluable selected equations only; the counts pool those equations. Methods have different denominators, so this table is a conditional diagnostic rather than a standalone success ranking. The true-variable set comes from the official reindexed ground-truth expression.

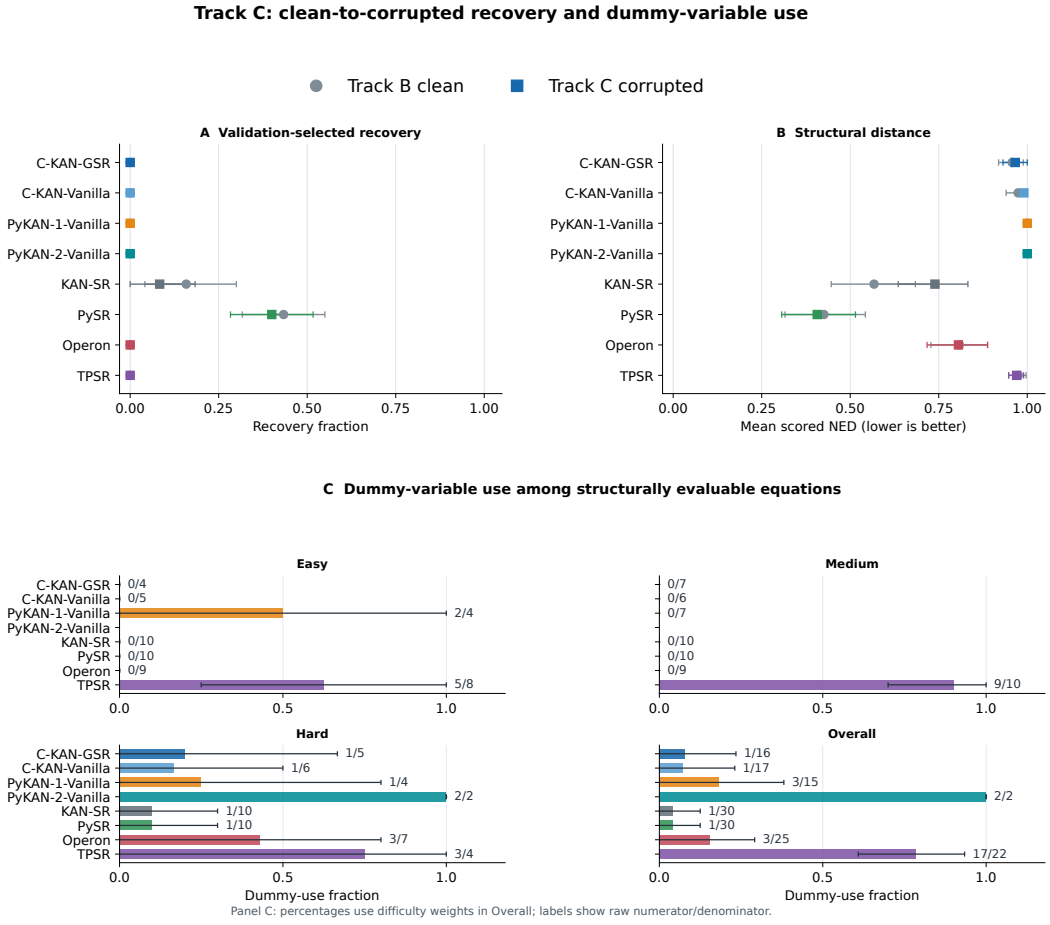

Figure 7: Provisional clean-to-corrupted summaries for the eight displayed outcomes, all with complete Track B and Track C seed blocks, plus conditional dummy-variable use. Circles mark clean Track B values and squares the corrupted Track C values. Where only one marker is visible, as for C-KAN, Operon and TPSR recovery, the two coincide. The incomplete PyKAN-GSR runs are excluded. The Track C corrupted-input hashes still require reconciliation before cross-method inference.

Table 26: Provisional Track C and paired Track B validation-selected outcomes by equation

Difficulty	Equation	Method	Z_B	Z_C	NED$_B$	NED$_C$	Clean R^2_C	Dummy
Easy	feynman-i.12.1	C-KAN-GSR	0	0	1.000	1.000	1.000	—
Easy	feynman-i.12.1	C-KAN-Vanilla	0	0	1.000	1.000	0.999	0
Easy	feynman-i.12.1	PyKAN-1-Vanilla	0	0	1.000	1.000	0.990	—
Easy	feynman-i.12.1	PyKAN-2-Vanilla	0	0	1.000	1.000	0.681	—
Easy	feynman-i.12.1	KAN-SR	0	0	0.333	0.333	1.000	0
Easy	feynman-i.12.1	PySR	1	1	0.000	0.000	1.000	0
Easy	feynman-i.12.1	Operon	0	0	1.000	1.000	1.000	0
Easy	feynman-i.12.1	TPSR	0	0	1.000	1.000	0.965	1
Easy	feynman-i.14.4	C-KAN-GSR	0	0	1.000	1.000	1.000	—
Easy	feynman-i.14.4	C-KAN-Vanilla	0	0	1.000	1.000	1.000	—
Easy	feynman-i.14.4	PyKAN-1-Vanilla	0	0	1.000	1.000	0.946	0
Easy	feynman-i.14.4	PyKAN-2-Vanilla	0	0	1.000	1.000	0.212	—
Easy	feynman-i.14.4	KAN-SR	1	1	0.000	0.000	1.000	0
Easy	feynman-i.14.4	PySR	1	1	0.000	0.000	1.000	0
Easy	feynman-i.14.4	Operon	0	0	0.333	0.333	1.000	0
Easy	feynman-i.14.4	TPSR	0	0	1.000	1.000	0.139	1
Easy	feynman-i.18.16	C-KAN-GSR	0	0	1.000	1.000	0.985	—
Easy	feynman-i.18.16	C-KAN-Vanilla	0	0	1.000	1.000	0.982	—
Easy	feynman-i.18.16	PyKAN-1-Vanilla	0	0	1.000	1.000	-0.708	—
Easy	feynman-i.18.16	PyKAN-2-Vanilla	0	0	1.000	1.000	-3.274	—
Easy	feynman-i.18.16	KAN-SR	0	0	0.500	1.000	1.000	0
Easy	feynman-i.18.16	PySR	1	1	0.000	0.000	1.000	0
Easy	feynman-i.18.16	Operon	0	0	0.500	0.833	1.000	0
Easy	feynman-i.18.16	TPSR	0	0	1.000	1.000	—	—
Easy	feynman-i.25.13	C-KAN-GSR	0	0	1.000	1.000	0.981	0
Easy	feynman-i.25.13	C-KAN-Vanilla	0	0	1.000	1.000	0.973	0
Easy	feynman-i.25.13	PyKAN-1-Vanilla	0	0	1.000	1.000	0.481	—
Easy	feynman-i.25.13	PyKAN-2-Vanilla	0	0	1.000	1.000	-0.270	—
Easy	feynman-i.25.13	KAN-SR	0	0	0.200	0.200	1.000	0
Easy	feynman-i.25.13	PySR	1	1	0.000	0.000	1.000	0
Easy	feynman-i.25.13	Operon	0	0	0.400	0.600	1.000	0
Easy	feynman-i.25.13	TPSR	0	0	1.000	1.000	-0.000	0
Easy	feynman-i.27.6	C-KAN-GSR	0	0	1.000	1.000	0.976	—
Easy	feynman-i.27.6	C-KAN-Vanilla	0	0	1.000	1.000	0.916	—
Easy	feynman-i.27.6	PyKAN-1-Vanilla	0	0	1.000	1.000	0.860	—
Easy	feynman-i.27.6	PyKAN-2-Vanilla	0	0	1.000	1.000	0.558	—
Easy	feynman-i.27.6	KAN-SR	0	0	0.200	1.000	0.936	0
Easy	feynman-i.27.6	PySR	0	1	1.000	0.000	1.000	0
Easy	feynman-i.27.6	Operon	0	0	1.000	1.000	1.000	0
Easy	feynman-i.27.6	TPSR	0	0	1.000	1.000	0.218	0
Easy	feynman-i.47.23	C-KAN-GSR	0	0	1.000	1.000	0.999	0
Easy	feynman-i.47.23	C-KAN-Vanilla	0	0	1.000	1.000	0.993	0
Easy	feynman-i.47.23	PyKAN-1-Vanilla	0	0	1.000	1.000	0.993	1
Easy	feynman-i.47.23	PyKAN-2-Vanilla	0	0	1.000	1.000	0.950	—
Easy	feynman-i.47.23	KAN-SR	0	0	1.000	1.000	1.000	0

Difficulty	Equation	Method	Z_B	Z_C	NED$_B$	NED$_C$	Clean R_C^2	Dummy
Easy	feynman-i.47.23	PySR	1	1	0.000	0.000	1.000	0
Easy	feynman-i.47.23	Operon	0	0	0.500	1.000	1.000	0
Easy	feynman-i.47.23	TPSR	0	0	1.000	1.000	0.233	1
Easy	feynman-ii.2.42	C-KAN-GSR	0	0	1.000	1.000	0.665	—
Easy	feynman-ii.2.42	C-KAN-Vanilla	0	0	1.000	1.000	0.097	0
Easy	feynman-ii.2.42	PyKAN-1-Vanilla	0	0	1.000	1.000	0.388	—
Easy	feynman-ii.2.42	PyKAN-2-Vanilla	0	0	1.000	1.000	0.444	—
Easy	feynman-ii.2.42	KAN-SR	0	0	0.091	1.000	0.208	0
Easy	feynman-ii.2.42	PySR	1	0	0.000	0.273	1.000	0
Easy	feynman-ii.2.42	Operon	0	0	1.000	1.000	1.000	0
Easy	feynman-ii.2.42	TPSR	0	0	1.000	0.818	0.003	0
Easy	feynman-ii.8.31	C-KAN-GSR	0	0	1.000	1.000	1.000	0
Easy	feynman-ii.8.31	C-KAN-Vanilla	0	0	1.000	1.000	1.000	0
Easy	feynman-ii.8.31	PyKAN-1-Vanilla	0	0	1.000	1.000	0.944	1
Easy	feynman-ii.8.31	PyKAN-2-Vanilla	0	0	1.000	1.000	0.781	—
Easy	feynman-ii.8.31	KAN-SR	1	1	0.000	0.000	1.000	0
Easy	feynman-ii.8.31	PySR	1	1	0.000	0.000	1.000	0
Easy	feynman-ii.8.31	Operon	0	0	0.400	0.400	1.000	0
Easy	feynman-ii.8.31	TPSR	0	0	1.000	1.000	0.238	1
Easy	feynman-ii.15.4	C-KAN-GSR	0	0	1.000	1.000	0.647	—
Easy	feynman-ii.15.4	C-KAN-Vanilla	0	0	1.000	1.000	0.059	—
Easy	feynman-ii.15.4	PyKAN-1-Vanilla	0	0	1.000	1.000	0.649	—
Easy	feynman-ii.15.4	PyKAN-2-Vanilla	0	0	1.000	1.000	—	—
Easy	feynman-ii.15.4	KAN-SR	0	0	0.833	0.333	1.000	0
Easy	feynman-ii.15.4	PySR	0	0	0.667	0.833	1.000	0
Easy	feynman-ii.15.4	Operon	0	0	1.000	1.000	—	—
Easy	feynman-ii.15.4	TPSR	0	0	1.000	1.000	-9.96×10^{32}	—
Easy	feynman-ii.38.14	C-KAN-GSR	0	0	1.000	1.000	1.000	0
Easy	feynman-ii.38.14	C-KAN-Vanilla	0	0	1.000	1.000	0.999	—
Easy	feynman-ii.38.14	PyKAN-1-Vanilla	0	0	1.000	1.000	0.995	0
Easy	feynman-ii.38.14	PyKAN-2-Vanilla	0	0	1.000	1.000	0.621	—
Easy	feynman-ii.38.14	KAN-SR	0	0	0.250	0.500	1.000	0
Easy	feynman-ii.38.14	PySR	1	1	0.000	0.000	1.000	0
Easy	feynman-ii.38.14	Operon	0	0	1.000	1.000	1.000	0
Easy	feynman-ii.38.14	TPSR	0	0	1.000	1.000	0.998	1
Medium	feynman-i.8.14	C-KAN-GSR	0	0	1.000	1.000	0.961	0
Medium	feynman-i.8.14	C-KAN-Vanilla	0	0	1.000	1.000	0.960	0
Medium	feynman-i.8.14	PyKAN-1-Vanilla	0	0	1.000	1.000	0.704	0
Medium	feynman-i.8.14	PyKAN-2-Vanilla	0	0	1.000	1.000	0.085	—
Medium	feynman-i.8.14	KAN-SR	0	0	0.870	1.000	0.796	0
Medium	feynman-i.8.14	PySR	1	1	0.000	0.000	1.000	0
Medium	feynman-i.8.14	Operon	0	0	1.000	1.000	0.996	0
Medium	feynman-i.8.14	TPSR	0	0	0.870	0.913	0.294	1
Medium	feynman-i.11.19	C-KAN-GSR	0	0	1.000	1.000	0.276	0
Medium	feynman-i.11.19	C-KAN-Vanilla	0	0	1.000	1.000	0.267	0
Medium	feynman-i.11.19	PyKAN-1-Vanilla	0	0	1.000	1.000	0.992	—
Medium	feynman-i.11.19	PyKAN-2-Vanilla	0	0	1.000	1.000	—	—

Difficulty	Equation	Method	Z_B	Z_C	NED$_B$	NED$_C$	Clean R^2_C	Dummy
Medium	feynman-i.11.19	KAN-SR	1	0	0.000	1.000	-0.013	0
Medium	feynman-i.11.19	PySR	1	1	0.000	0.000	1.000	0
Medium	feynman-i.11.19	Operon	0	0	0.900	0.400	1.000	0
Medium	feynman-i.11.19	TPSR	0	0	1.000	1.000	-0.000	1
Medium	feynman-i.12.11	C-KAN-GSR	0	0	1.000	1.000	0.724	0
Medium	feynman-i.12.11	C-KAN-Vanilla	0	0	1.000	1.000	0.725	0
Medium	feynman-i.12.11	PyKAN-1-Vanilla	0	0	1.000	1.000	0.401	—
Medium	feynman-i.12.11	PyKAN-2-Vanilla	0	0	1.000	1.000	-3.552	—
Medium	feynman-i.12.11	KAN-SR	0	0	0.222	1.000	0.924	0
Medium	feynman-i.12.11	PySR	1	1	0.000	0.000	1.000	0
Medium	feynman-i.12.11	Operon	0	0	0.778	1.000	1.000	0
Medium	feynman-i.12.11	TPSR	0	0	1.000	1.000	-0.001	1
Medium	feynman-i.16.6	C-KAN-GSR	0	0	0.583	1.000	1.000	0
Medium	feynman-i.16.6	C-KAN-Vanilla	0	0	1.000	1.000	0.999	0
Medium	feynman-i.16.6	PyKAN-1-Vanilla	0	0	1.000	1.000	0.996	0
Medium	feynman-i.16.6	PyKAN-2-Vanilla	0	0	1.000	1.000	0.928	—
Medium	feynman-i.16.6	KAN-SR	0	0	0.667	0.667	0.999	0
Medium	feynman-i.16.6	PySR	0	0	0.750	0.750	0.999	0
Medium	feynman-i.16.6	Operon	0	0	1.000	1.000	1.000	0
Medium	feynman-i.16.6	TPSR	0	0	1.000	1.000	-0.000	1
Medium	feynman-i.18.4	C-KAN-GSR	0	0	1.000	1.000	0.842	0
Medium	feynman-i.18.4	C-KAN-Vanilla	0	0	1.000	1.000	0.892	—
Medium	feynman-i.18.4	PyKAN-1-Vanilla	0	0	1.000	1.000	0.992	—
Medium	feynman-i.18.4	PyKAN-2-Vanilla	0	0	1.000	1.000	0.608	—
Medium	feynman-i.18.4	KAN-SR	0	0	1.000	1.000	0.859	0
Medium	feynman-i.18.4	PySR	1	1	0.000	0.000	1.000	0
Medium	feynman-i.18.4	Operon	0	0	1.000	1.000	0.998	0
Medium	feynman-i.18.4	TPSR	0	0	1.000	1.000	-0.001	1
Medium	feynman-i.29.4	C-KAN-GSR	0	0	0.667	0.667	1.000	0
Medium	feynman-i.29.4	C-KAN-Vanilla	0	0	0.667	1.000	1.000	0
Medium	feynman-i.29.4	PyKAN-1-Vanilla	0	0	1.000	1.000	0.934	0
Medium	feynman-i.29.4	PyKAN-2-Vanilla	0	0	1.000	1.000	0.718	—
Medium	feynman-i.29.4	KAN-SR	1	1	0.000	0.000	1.000	0
Medium	feynman-i.29.4	PySR	1	1	0.000	0.000	1.000	0
Medium	feynman-i.29.4	Operon	0	0	0.667	0.667	1.000	0
Medium	feynman-i.29.4	TPSR	0	0	1.000	1.000	0.268	1
Medium	feynman-ii.6.11	C-KAN-GSR	0	0	1.000	1.000	0.203	0
Medium	feynman-ii.6.11	C-KAN-Vanilla	0	0	1.000	1.000	0.207	0
Medium	feynman-ii.6.11	PyKAN-1-Vanilla	0	0	1.000	1.000	-0.634	0
Medium	feynman-ii.6.11	PyKAN-2-Vanilla	0	0	1.000	1.000	—	—
Medium	feynman-ii.6.11	KAN-SR	0	0	0.250	1.000	1.000	0
Medium	feynman-ii.6.11	PySR	1	1	0.000	0.000	1.000	0
Medium	feynman-ii.6.11	Operon	0	0	0.500	0.500	1.000	0
Medium	feynman-ii.6.11	TPSR	0	0	0.750	1.000	-0.000	1
Medium	feynman-iii.4.32	C-KAN-GSR	0	0	1.000	1.000	0.992	—
Medium	feynman-iii.4.32	C-KAN-Vanilla	0	0	1.000	1.000	0.476	—
Medium	feynman-iii.4.32	PyKAN-1-Vanilla	0	0	1.000	1.000	0.019	0

Difficulty	Equation	Method	Z_B	Z_C	NED$_B$	NED$_C$	Clean R^2_C	Dummy
Medium	feynman-iii.4.32	PyKAN-2-Vanilla	0	0	1.000	1.000	0.314	—
Medium	feynman-iii.4.32	KAN-SR	0	0	0.636	0.636	1.000	0
Medium	feynman-iii.4.32	PySR	0	0	1.000	0.636	1.000	0
Medium	feynman-iii.4.32	Operon	0	0	0.455	0.455	1.000	0
Medium	feynman-iii.4.32	TPSR	0	0	1.000	1.000	-0.001	0
Medium	feynman-iii.15.12	C-KAN-GSR	0	0	1.000	1.000	0.991	—
Medium	feynman-iii.15.12	C-KAN-Vanilla	0	0	1.000	1.000	0.990	—
Medium	feynman-iii.15.12	PyKAN-1-Vanilla	0	0	1.000	1.000	0.053	0
Medium	feynman-iii.15.12	PyKAN-2-Vanilla	0	0	1.000	1.000	0.627	—
Medium	feynman-iii.15.12	KAN-SR	0	0	1.000	0.818	0.992	0
Medium	feynman-iii.15.12	PySR	0	0	1.000	0.909	1.000	0
Medium	feynman-iii.15.12	Operon	0	0	1.000	1.000	—	—
Medium	feynman-iii.15.12	TPSR	0	0	1.000	0.818	-1.53×10^{28}	1
Medium	feynman-bonus.18	C-KAN-GSR	0	0	1.000	1.000	0.998	—
Medium	feynman-bonus.18	C-KAN-Vanilla	0	0	1.000	1.000	0.949	—
Medium	feynman-bonus.18	PyKAN-1-Vanilla	0	0	1.000	1.000	0.577	0
Medium	feynman-bonus.18	PyKAN-2-Vanilla	0	0	1.000	1.000	0.101	—
Medium	feynman-bonus.18	KAN-SR	0	0	1.000	1.000	0.999	0
Medium	feynman-bonus.18	PySR	1	0	0.000	1.000	0.999	0
Medium	feynman-bonus.18	Operon	0	0	1.000	0.833	0.931	0
Medium	feynman-bonus.18	TPSR	0	0	1.000	1.000	-0.173	1
Hard	feynman-i.6.20a	C-KAN-GSR	0	0	1.000	1.000	1.000	1
Hard	feynman-i.6.20a	C-KAN-Vanilla	0	0	1.000	1.000	1.000	0
Hard	feynman-i.6.20a	PyKAN-1-Vanilla	0	0	1.000	1.000	-0.398	—
Hard	feynman-i.6.20a	PyKAN-2-Vanilla	0	0	1.000	1.000	-11.499	1
Hard	feynman-i.6.20a	KAN-SR	1	0	0.000	0.500	0.093	0
Hard	feynman-i.6.20a	PySR	0	0	0.250	0.500	1.000	0
Hard	feynman-i.6.20a	Operon	0	0	0.875	0.250	1.000	0
Hard	feynman-i.6.20a	TPSR	0	0	1.000	0.875	-0.006	1
Hard	feynman-i.30.3	C-KAN-GSR	0	0	0.833	0.722	-0.000	0
Hard	feynman-i.30.3	C-KAN-Vanilla	0	0	0.722	0.889	-0.001	0
Hard	feynman-i.30.3	PyKAN-1-Vanilla	0	0	1.000	1.000	-0.113	1
Hard	feynman-i.30.3	PyKAN-2-Vanilla	0	0	1.000	1.000	-0.015	—
Hard	feynman-i.30.3	KAN-SR	0	0	0.611	1.000	-0.996	1
Hard	feynman-i.30.3	PySR	0	0	0.500	0.667	-0.001	1
Hard	feynman-i.30.3	Operon	0	0	1.000	1.000	—	—
Hard	feynman-i.30.3	TPSR	0	0	0.778	0.889	-0.001	0
Hard	feynman-i.44.4	C-KAN-GSR	0	0	1.000	1.000	0.983	—
Hard	feynman-i.44.4	C-KAN-Vanilla	0	0	1.000	1.000	0.798	—
Hard	feynman-i.44.4	PyKAN-1-Vanilla	0	0	1.000	1.000	0.403	—
Hard	feynman-i.44.4	PyKAN-2-Vanilla	0	0	1.000	1.000	-4.129	—
Hard	feynman-i.44.4	KAN-SR	0	0	1.000	1.000	1.000	0
Hard	feynman-i.44.4	PySR	0	0	0.900	0.500	0.712	0
Hard	feynman-i.44.4	Operon	0	0	0.400	0.800	1.000	0
Hard	feynman-i.44.4	TPSR	0	0	1.000	1.000	-1.10×10^{16}	1
Hard	feynman-ii.13.23	C-KAN-GSR	0	0	1.000	1.000	1.000	0
Hard	feynman-ii.13.23	C-KAN-Vanilla	0	0	1.000	1.000	1.000	0

Difficulty	Equation	Method	Z_B	Z_C	NED$_B$	NED$_C$	Clean R^2_C	Dummy
Hard	feynman-ii.13.23	PyKAN-1-Vanilla	0	0	1.000	1.000	0.924	0
Hard	feynman-ii.13.23	PyKAN-2-Vanilla	0	0	1.000	1.000	0.686	1
Hard	feynman-ii.13.23	KAN-SR	0	0	0.818	0.818	1.000	0
Hard	feynman-ii.13.23	PySR	0	0	0.727	0.364	1.000	0
Hard	feynman-ii.13.23	Operon	0	0	0.636	0.636	-0.003	1
Hard	feynman-ii.13.23	TPSR	0	0	1.000	1.000	—	—
Hard	feynman-ii.35.21	C-KAN-GSR	0	0	1.000	1.000	0.640	—
Hard	feynman-ii.35.21	C-KAN-Vanilla	0	0	1.000	1.000	0.450	—
Hard	feynman-ii.35.21	PyKAN-1-Vanilla	0	0	1.000	1.000	0.918	—
Hard	feynman-ii.35.21	PyKAN-2-Vanilla	0	0	1.000	1.000	0.230	—
Hard	feynman-ii.35.21	KAN-SR	0	0	0.636	0.636	1.000	0
Hard	feynman-ii.35.21	PySR	0	0	0.909	1.000	0.792	0
Hard	feynman-ii.35.21	Operon	0	0	0.727	0.727	1.000	0
Hard	feynman-ii.35.21	TPSR	0	0	1.000	1.000	—	—
Hard	feynman-iii.9.52	C-KAN-GSR	0	0	1.000	1.000	1.000	—
Hard	feynman-iii.9.52	C-KAN-Vanilla	0	0	1.000	1.000	0.003	—
Hard	feynman-iii.9.52	PyKAN-1-Vanilla	0	0	1.000	1.000	0.859	—
Hard	feynman-iii.9.52	PyKAN-2-Vanilla	0	0	1.000	1.000	0.176	—
Hard	feynman-iii.9.52	KAN-SR	0	0	0.640	0.640	1.000	0
Hard	feynman-iii.9.52	PySR	0	0	0.800	0.800	0.343	0
Hard	feynman-iii.9.52	Operon	0	0	1.000	1.000	—	—
Hard	feynman-iii.9.52	TPSR	0	0	1.000	0.840	-0.004	1
Hard	feynman-bonus.6	C-KAN-GSR	0	0	0.875	1.000	0.102	0
Hard	feynman-bonus.6	C-KAN-Vanilla	0	0	1.000	1.000	0.112	1
Hard	feynman-bonus.6	PyKAN-1-Vanilla	0	0	1.000	1.000	-2.676	—
Hard	feynman-bonus.6	PyKAN-2-Vanilla	0	0	1.000	1.000	0.032	—
Hard	feynman-bonus.6	KAN-SR	0	0	1.000	0.958	0.076	0
Hard	feynman-bonus.6	PySR	0	0	0.750	0.500	0.011	0
Hard	feynman-bonus.6	Operon	0	0	1.000	1.000	0.453	1
Hard	feynman-bonus.6	TPSR	0	0	1.000	1.000	-0.017	—
Hard	feynman-bonus.12	C-KAN-GSR	0	0	0.865	0.731	-0.001	0
Hard	feynman-bonus.12	C-KAN-Vanilla	0	0	0.923	0.885	-0.001	0
Hard	feynman-bonus.12	PyKAN-1-Vanilla	0	0	1.000	1.000	-0.050	—
Hard	feynman-bonus.12	PyKAN-2-Vanilla	0	0	1.000	1.000	-0.093	—
Hard	feynman-bonus.12	KAN-SR	0	0	0.846	0.827	-0.000	0
Hard	feynman-bonus.12	PySR	0	0	0.865	0.769	-561.932	0
Hard	feynman-bonus.12	Operon	0	0	1.000	0.865	-0.228	1
Hard	feynman-bonus.12	TPSR	0	0	0.923	1.000	-1.02×10^{99}	—
Hard	feynman-bonus.17	C-KAN-GSR	0	0	1.000	1.000	0.995	—
Hard	feynman-bonus.17	C-KAN-Vanilla	0	0	1.000	1.000	0.975	0
Hard	feynman-bonus.17	PyKAN-1-Vanilla	0	0	1.000	1.000	-1.043	0
Hard	feynman-bonus.17	PyKAN-2-Vanilla	0	0	1.000	1.000	-0.294	—
Hard	feynman-bonus.17	KAN-SR	0	0	0.742	0.742	1.000	0
Hard	feynman-bonus.17	PySR	0	0	0.742	0.742	1.000	0
Hard	feynman-bonus.17	Operon	0	0	0.839	0.871	0.002	0
Hard	feynman-bonus.17	TPSR	0	0	1.000	1.000	-0.099	—
Hard	feynman-bonus.20	C-KAN-GSR	0	0	1.000	1.000	1.000	—

Difficulty	Equation	Method	Z_B	Z_C	NED$_B$	NED$_C$	Clean R^2_C	Dummy
Hard	feynman-bonus.20	C-KAN-Vanilla	0	0	1.000	1.000	0.963	—
Hard	feynman-bonus.20	PyKAN-1-Vanilla	0	0	1.000	1.000	-1.320	0
Hard	feynman-bonus.20	PyKAN-2-Vanilla	0	0	1.000	1.000	0.053	—
Hard	feynman-bonus.20	KAN-SR	0	0	0.762	0.905	0.997	0
Hard	feynman-bonus.20	PySR	0	0	0.571	0.619	0.866	0
Hard	feynman-bonus.20	Operon	0	0	1.000	1.000	—	—
Hard	feynman-bonus.20	TPSR	0	0	1.000	1.000	—	—

R POST-HOC CLEAN-UP AND PRE-SYMBOLIC DIAGNOSTICS

This appendix reports diagnostics that were chosen after the benchmark results were seen. They are not part of the prespecified protocol, and the unprocessed results in Sections N.2 and N.3 remain the primary outcomes. No ledger was modified. The scripts are in `posthoc_kansr_cleanup/` (`posthoc.py`, `presymbolic.py` and `taxonomy.py`).

Procedure. For every production attempt that each track's analyzer selects, the returned original-scale equation is post-processed using that attempt's training data only. Two variants follow KAN-SR (Bühler & Guillén-Gosálbez, 2025):

- *Rounding and refit.* Inputs and target are divided by their training standard deviation without centering, following the benchmark's KAN-SR wrapper. Integer powers of sums are multiplied out unless this more than triples the expression. Every constant is rounded to two decimals, as in KAN-SR's final rounding. The surviving non-exponent constants are then refit by least squares, and the result is mapped back to original units.

- *Term elimination, then rounding and refit.* This uses KAN-SR's optional backward elimination. Each constant in turn is set to zero, the rest are refit, and the best removal is accepted if the training MSE rises by at most 1%. The loop repeats until no removal qualifies. It is limited to 30 constants, 1,000 training rows and 300 seconds per attempt. `sympy.simplify` is omitted inside the loop for time.

A clean-up that fails, times out, produces an undefined expression, or removes every variable returns its input unchanged. The last case arises with heavy-tailed targets: after dividing by the target's standard deviation, typical values are small, and two-decimal rounding can set every coefficient to zero. Each result is then scored with the track's own code: the SRSD canonicalizer and unit-cost tree distance with the frozen 120-second allowance, the official pointwise validation MSRE, and clean test R^2. The track's validation-based seed selection, difficulty weights and equation bootstrap are then re-run. Re-scoring the unmodified expressions reproduced the recorded NED values.

In Track B, all methods used the official SRSD files, so all seven methods (both C-KAN modes, PyKAN-1-Vanilla, PySR, Operon, KAN-SR and TPSR) are post-processed identically. PyKAN-1 was added after its runs were collected, using the same code. In Track C, only C-KAN's corrupted inputs are stored locally, one run folder per paired evolution. The baselines' corrupted inputs were generated on other machines, so the baselines cannot be re-scored.

Table 27: Post-hoc clean-up on Track B (clean data). SR^{val} is validation-selected skeleton recovery, difficulty-weighted, with its 95% equation-bootstrap interval. SR^{any} is the any-seed oracle. Size is the median ratio of simplified size to ground-truth size among evaluable selections. Rows marked R+R apply rounding and refit; rows marked E+R+R apply term elimination, then rounding and refit.

Method	Variant	SR^{val} (%)	SR^{any} (%)	Mean NED	$R^2 > 0.999$ (%)	Evaluable (%)	Size ratio
C-KAN-GSR	recorded	0.0 [0.0, 0.0]	0.0	0.957	37.5	43.3	7.50
	R+R	5.8 [0.0, 15.0]	5.8	0.928	45.0	49.2	5.10
	E+R+R	5.8 [0.0, 15.0]	8.3	0.873	40.8	70.0	2.00
C-KAN-Vanilla	recorded	0.0 [0.0, 0.0]	0.0	0.974	25.0	45.0	3.29
	R+R	3.3 [0.0, 10.0]	3.3	0.927	30.8	62.5	2.30
	E+R+R	5.8 [0.0, 15.0]	5.8	0.888	33.3	67.5	2.67
PyKAN-1-Vanilla	recorded	0.0 [0.0, 0.0]	0.0	1.000	0.0	50.0	14.60
	R+R	0.0 [0.0, 0.0]	0.0	1.000	54.2	46.7	14.58
	E+R+R	5.8 [0.0, 15.0]	5.8	0.942	50.0	44.2	7.39
PySR	recorded	43.3 [31.7, 55.0]	45.8	0.426	71.7	96.7	1.00
	R+R	27.5 [15.8, 40.0]	30.8	0.451	68.3	96.7	1.00
	E+R+R	27.5 [15.8, 40.0]	30.8	0.451	68.3	96.7	1.00
Operon	recorded	0.0 [0.0, 0.0]	0.0	0.808	55.0	74.2	1.50
	R+R	24.2 [13.3, 35.8]	28.3	0.632	55.8	70.8	1.00
	E+R+R	24.2 [13.3, 35.8]	28.3	0.639	55.8	70.8	1.00
KAN-SR	recorded	15.8 [4.2, 30.0]	18.3	0.567	77.5	96.7	1.04
	R+R	30.8 [18.3, 45.0]	30.8	0.541	77.5	96.7	1.00
	E+R+R	30.8 [18.3, 45.0]	30.8	0.530	77.5	96.7	1.00
TPSR	recorded	0.0 [0.0, 0.0]	0.0	0.975	0.0	85.0	3.20
	R+R	0.0 [0.0, 0.0]	0.0	0.950	6.7	78.3	2.50
	E+R+R	9.2 [0.0, 20.0]	9.2	0.834	15.0	81.7	1.00

Table 28: Post-hoc clean-up of C-KAN on Track C (corrupted data; provisional, see Section N.3). Columns as in Table 27.

Method	Variant	SR^{val} (%)	SR^{any} (%)	Mean NED	$R^2 > 0.999$ (%)	Evaluable (%)	Size ratio
C-KAN-GSR	recorded	0.0 [0.0, 0.0]	0.0	0.966	35.8	54.2	2.77
	R+R	0.0 [0.0, 0.0]	5.8	0.954	35.8	64.2	2.62
	E+R+R	3.3 [0.0, 10.0]	5.8	0.921	35.8	58.3	1.36
C-KAN-Vanilla	recorded	0.0 [0.0, 0.0]	0.0	0.991	20.0	57.5	3.12
	R+R	3.3 [0.0, 10.0]	3.3	0.958	26.7	69.2	2.88
	E+R+R	0.0 [0.0, 0.0]	5.8	0.957	20.0	75.0	2.38

Reading the tables.

- *C-KAN in Track B.* Clean-up recovers the two residue cases, `ii.8.31` and `i.29.4`. Term elimination recovers `i.14.4` in one additional seed, but that seed is not the one chosen by validation.
- *PyKAN-1.* Its returned equations often have a usable form but poorly fitted constants: the median test R^2 of its 91 re-scorable runs is 0.78 as recorded and 0.9998 after the least-squares refit. Its high-accuracy rate therefore rises from 0 to 54.2%, above C-KAN-GSR under the same treatment. Its equations stay larger (size ratio 7.4 against 2.0 after term elimination) and structurally further from the target (mean NED 0.942 against 0.873). PyKAN's own post-pruning refit diverged in 85 of 150 runs, and the pipeline then kept the unrefined pruned network.
- *Operon and KAN-SR.* Their large gains come from equations that were already correct up to tiny offsets, or up to coefficients that differ from one in the last digits.
- *PySR.* Its loss is the same effect in reverse. The canonicalizer replaces a constant only when it equals one exactly, so rescaling and refitting an exact, constant-free expression adds a constant node.
- *Other effects.* A cleaned equation can also become unevaluable, when it exceeds the canonicalization allowance or its validation loss becomes undefined. This accounts for the small drops in evaluability for Operon and TPSR.

Taken together, these results show that skeleton recovery on SRSD depends on constant formatting in both directions. A difference of a few points between methods should therefore be read cautiously.

Pre-symbolic accuracy. No stored file contains a pre-symbolic score on the official test set: the candidate field reserved for it is empty in every row, and no network state was saved. The stored pre-symbolic measures are MSEs of the standardized target on C-KAN's internal 10% training-only split. They cover the final-generation champion before pruning, each prepared candidate after pruning and refitting, and the same candidate after symbolic conversion. Table 29 counts how many reach NMSE below 10^{-3}. In Track C the fixed threshold is less meaningful, because the injected noise alone can exceed it when a target's mean is large relative to its spread. For this reason the stage breakdown in Table 21 uses Track B only.

Table 29: C-KAN-GSR models reaching internal-split NMSE below 10^{-3}, by pipeline stage.

	Champion before pruning	Best prepared network	Best symbolic candidate
Track B, runs (of 149 with telemetry)	34	46	42
Track B, equations (of 30, any seed)	9	13	12
Track C, runs (of 147 with telemetry)	20	38	39
Track C, equations (of 30, any seed)	6	8	10

S ADDITIONAL ABLATIONS

All results in this appendix come from the Track B ledgers, processed by `analyze_ablation.py` and rendered by `render_ablation.py` and `render_ablation_figures.py`. The only exception is the cross-track extraction comparison, produced by `analyze_ckan_gsr.py`. Selection and scoring follow Section 4.4. The component block has a nominal 5,000-second fit cap, but 8, 8, 3 and 10 of its 60 GSR attempts, respectively, exceeded that cap under the older controller; maxima were about 10,000 seconds. In the six-equation candidate-count study, 8 of 30 $K = 7$ attempts reached as much as 7,200 seconds, and 5 of 30 $K = 10$ attempts exceeded their nominal 5,000-second cap. These pilot rows remain descriptive and are not equal-compute evidence for a value of K. The later main-batch removals stayed within their 3,600-second fit caps.

Neural uncertainty and sparsity regularization. Table 30 adds two optional library features to Main on the six-equation block:

- *Neural uncertainty* perturbs inputs and internal activations with amplitude 0.02 during fitness evaluation.
- *Sparsity regularization* adds a PyKAN-style L1 penalty ($\lambda_{\mathrm{L1}} = 0.003$) to the gradient steps and a coefficient-difference penalty (3×10^{-6}) to fitness.

Neither changes accuracy or NED measurably, since all three rows sit at zero high-accuracy equations. Neural uncertainty lowered validity to 6 of 30 runs, from 10 of 30. Sparsity regularization left validity near Main's (11 of 30), but its selected equations were larger (median 40 nodes against 19).

The edge-fit column measures something different from test R^2: for each selected candidate, it is the mean R^2 of its symbolic edges' local fits to their spline curves. It shows no clear gain from either feature. Figure 8 shows that sparsity regularization does not shrink the active graph during evolution. Parsimony already acts through selection, and the regularized population reaches the same median active edge and node counts as Main. The sparsity run used different hardware from Main, so we do not compare their times.

Vanilla versus GSR extraction across all tracks. Every C-KAN evolution produces both a vanilla and a GSR equation from the same prepared candidates. We can therefore pair the two extractions exactly across Tracks A, B and C, regardless of dataset (Table 31). Over the 469 main-configuration pairs:

Table 30: Additional Track B C-KAN ablations. Edge-fit R^2 is the median [Q1, Q3] across selected candidates of their mean local symbolic-edge fit; it is not held-out equation R^2. Missing entries indicate that the corresponding run or operator-fit telemetry is unavailable.

Variant	n_d	SR (%)	Acc. (%)	NED ↓	Nodes	Valid (%)	Edge-fit R^2
6 equations (1 Medium, 5 Hard), 3600 s							
Main, $K = 5$ (main batch)	6/6	0.0%	0.0%	0.951	19.0	33.3%	0.904 [0.574, 0.934]
+neural uncertainty	6/6	0.0%	0.0%	0.954	33.0	20.0%	0.902 [0.805, 0.937]
+sparsity regularization	6/6	0.0%	0.0%	0.954	40.0	36.7%	0.931 [0.770, 0.979]

Table 31: Paired C-KAN extraction comparison from the same evolved population. Δ is GSR minus Vanilla; positive test R^2 and negative NED favor GSR. ΔR^2 uses pairs with two valid finite test predictions; structural failure is separate. Track A has no NED; failed SRSD equations receive NED 1. These are descriptive run-level comparisons; seeds and datasets are not independent replicates of one experiment. The exploratory row mixes pilot budgets and ablation configurations. Track rows use the main $K = 5$ configuration. Gain/loss counts exclude exact ties.

Track	Pairs	Both pred.	V only	G only	Med. ΔR^2	R^2 gain/loss	Med. ΔNED	NED gain/loss
Track A	139	100	4	4	0.00148	58/42	—	—
Track B	180	118	1	52	0.00043	79/30	0.00000	9/5
Track C	150	111	0	17	0.00034	76/29	0.00000	10/2
Pooled main $K = 5$	469	329	5	73	0.00055	213/101	0.00000	19/7
All variants (exploratory)	903	695	7	110	0.00113	450/185	0.00000	51/11

- *Validity.* GSR produced a valid prediction where vanilla did not in 73 pairs; the reverse happened in 5. Most of this asymmetry comes from the ordering. GSR is extracted first, and vanilla runs out of the shared deadline, so it is not evidence that GSR converts more reliably.
- *Accuracy.* Where both are valid, GSR's test R^2 is higher in 213 pairs and lower in 101. The median gain is small ($+0.00055$).
- *Structure.* Scored NED differs in only 26 SRSD pairs, 19 of them in GSR's favor.

The local edge fits show the design trade-off. In the component block, vanilla's selected candidates have median edge-fit R^2 0.989 [0.949, 0.995], against 0.932 [0.863, 0.985] for GSR. Vanilla picks each edge's best local match. GSR accepts slightly worse local fits when they improve the composed model's loss. The telemetry records the mean edge fit per candidate, not the per-operator median, so finer operator-level summaries are not available.

Search dynamics. Figure 8 shows per-generation population statistics for the component block, and for Main against sparsity regularization on the six-equation block.

- *Species.* Counts rise to a peak of about 17 near generation 25 and then decline to one or two by generation 100. The four SM and behavior-distance cells are nearly indistinguishable, which is why the main text reports this in one sentence.
- *Active structure.* Active edges and nodes grow during the first 30 generations and then plateau at a median of 13–16 edges and 8–9 nodes.
- *Offspring origin.* The composition shifts around generations 25–35, from mostly crossover to mostly cloning and back, as the number of species collapses. These curves show which operator produced each member of the population. They are not survival rates, which the telemetry does not record.

Variants not reported. Three planned variants lack a usable complete block:

- *Raw product nodes.* Their runs span two protocol versions with neither complete.
- *Imitation learning and no multiplicative nodes.* These were not collected.

The extractor lists all three as missing rather than pooling incompatible runs.

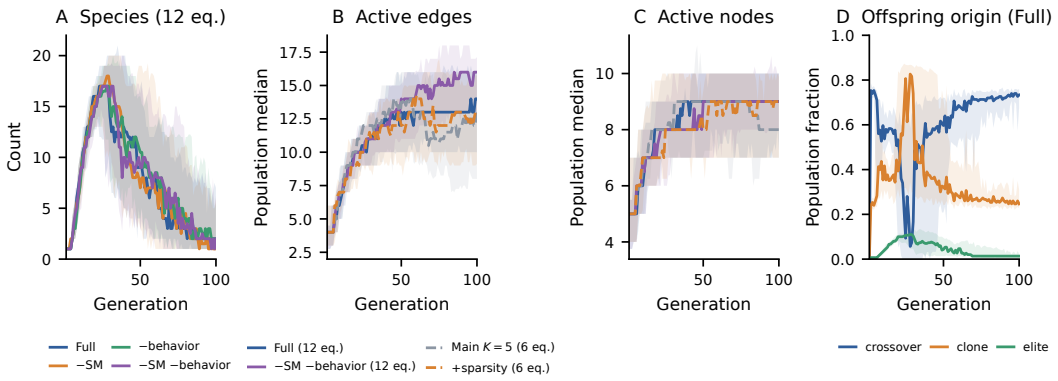

Figure 8: Search dynamics. Lines are medians across runs at each generation, and bands span the first to third quartiles. (A) Species count in the 2×2 component study. (B, C) Population-median active edges and nodes, for the two corners of the component study (12 equations) and for Main against sparsity regularization (six equations). (D) Fraction of the population produced by crossover, cloning or elitism in the full configuration.