vZero Neural ODE

Illustrated guide to the handwritten topics

A companion to the separate 35-slide explanation. Learn the rate, integrate the trajectory, and check the physics. The material distinguishes inspected vZero code, established methods, new teaching experiments and historical research evidence.

Suggested use: 35–45 minutes plus notebook exploration. This pacing is an assumption. Your technical quiz and content review remain open before LaTeX/TikZ authoring.

The handwritten-page map

Page 1 → Arrhenius, function composition and Koopman: slides 3–6 and 17–21.

Page 2 → Snapshots, turbulence, DMD and structured mechanics: slides 22–27.

Page 3 → Robot state and rate: slides 25–27.

Page 4 → Flame measurements, the unconfirmed stereo-DIC-like note and tomography: slide 29.

Page 5 → Adjoints, multipliers, Neural ODE and graph structure: slides 15–16 and 28.

Page 6 → Low rank, quantization and the ONNX-like note: slides 23–24 and 32–33.

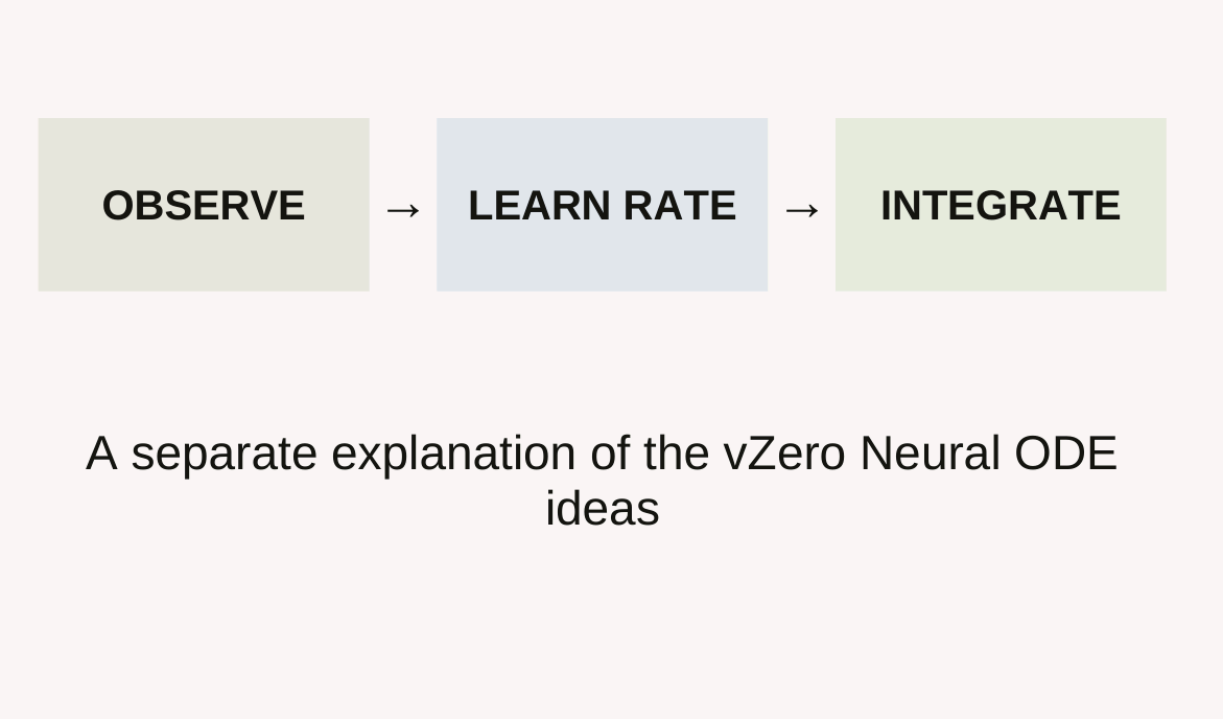
The isolated “75%” has no established denominator and is not used as a result. “ONX” is interpreted as ONNX in context; unclear handwriting remains explicitly unconfirmed.

THE MAP

1. vZero Neural ODE

From handwritten ideas to a working mental model

This presentation connects the handwritten topics to the vZero research workflow. It is separate from the conference paper on mechanism optimization. The audience needs only basic graphs, functions and the idea of a slope. We use three recurring analogies: microphones for competing training signals, knobs for modeling choices, and landscapes for optimization. Established methods, inspected package features, historical findings and new synthetic experiments are labeled separately. The purpose is to understand which component addresses which problem, not to claim that every method in the notes is already implemented.



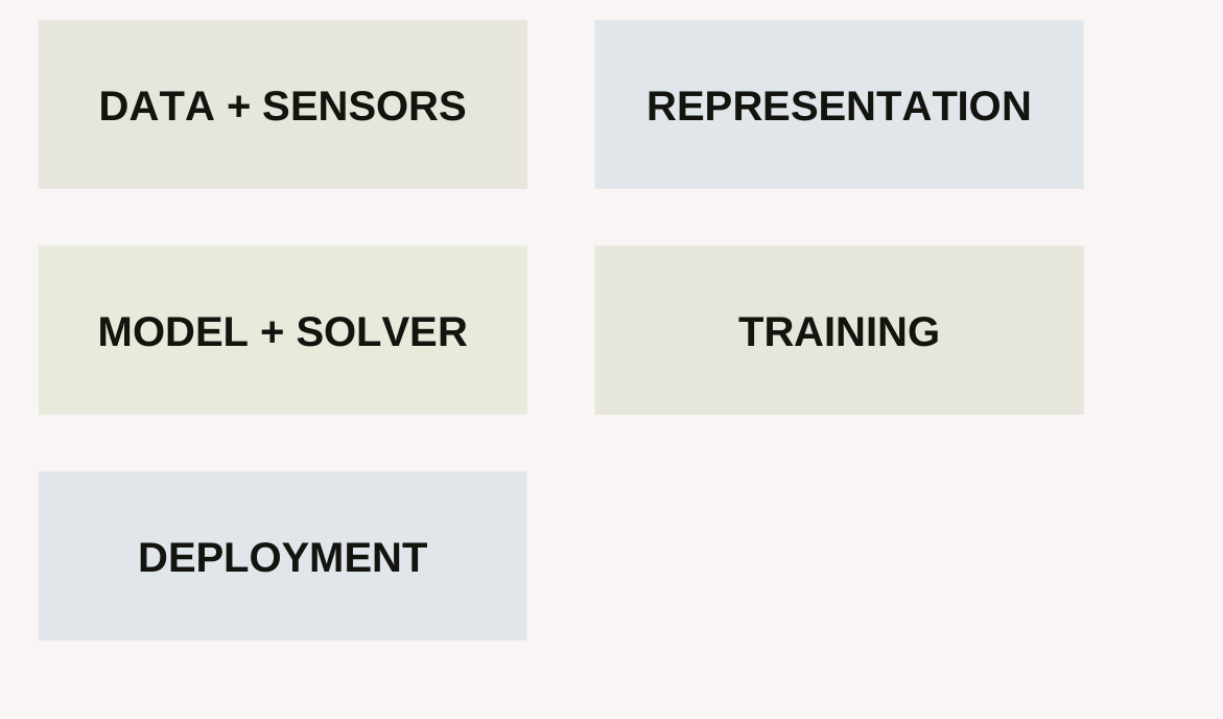
Sources: H1, V1. Full source details follow the glossary.

THE MAP

2. Five jobs, five different questions

A useful method belongs somewhere in this pipeline.

Organize the notes into five jobs. Data and the observation model explain what a measurement means. Representation describes the coordinates or features used for the state. A rate model and solver together generate time evolution. A loss and optimizer decide how parameters change during learning. Deployment specifies precision, runtime and hardware. Koopman lifting changes the representation and model; an adjoint calculates sensitivities; quantization changes numerical representation. Combining methods is meaningful when their inputs and outputs are compatible. Adding their names together is not yet an algorithm.



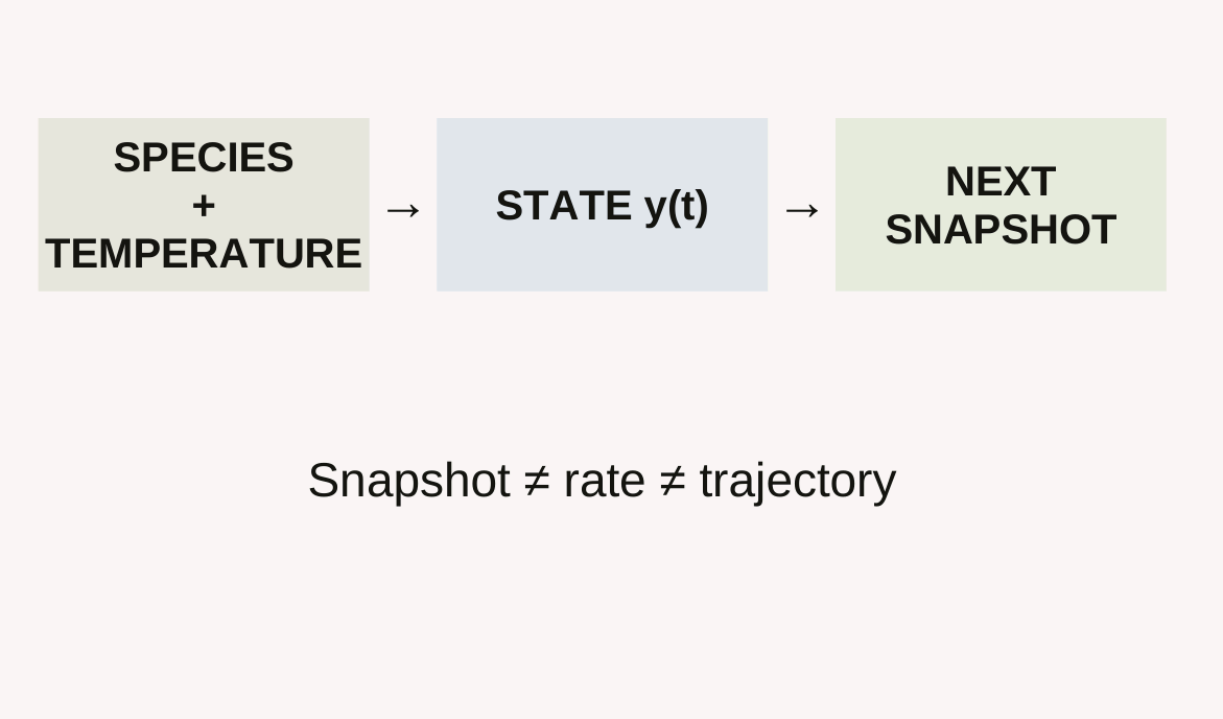
Sources: H1, V1. Full source details follow the glossary.

THE CORE

3. A state is a snapshot

A trajectory is a sequence of states indexed by time.

A state is the information needed by the chosen dynamical model to predict how the system changes. The documented H2/air example uses eight species mass fractions and temperature. This is a selected state description, not a claim that all chemical species are included or that these eight mass fractions sum to one. A robot arm instead needs positions and velocities; positions alone can be insufficient because the same pose can be moving in different directions. A camera image is an observation of a system and need not uniquely determine its full state.



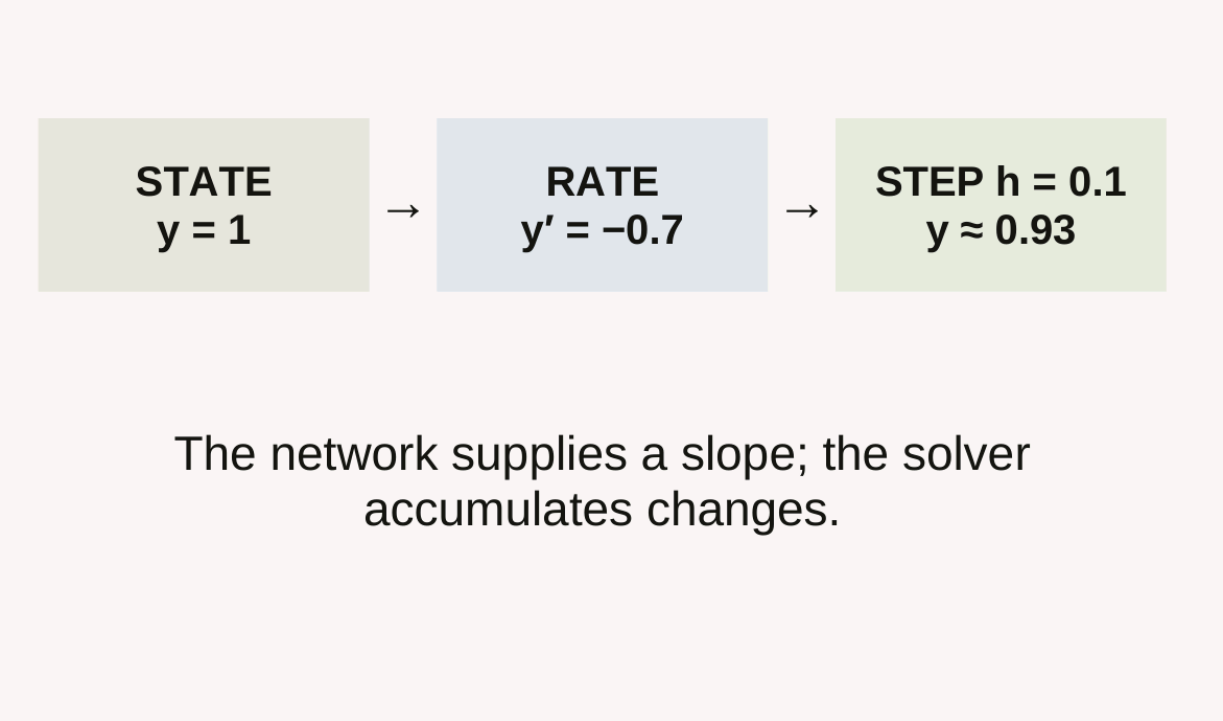
Sources: V3, H1. Full source details follow the glossary.

THE CORE

4. A rate is not the next state

The network supplies a slope; the solver accumulates changes.

For the toy decay $y' = -0.7y$, starting at $y=1$ gives an initial rate of -0.7 per unit time. A short Euler step of 0.1 predicts 0.93; the exact value is $\exp(-0.07)$, about 0.9324. The difference is numerical integration error. A Neural ODE uses a learned function to supply the rate at each solver evaluation. A direct next-state network is a different model: its output already represents a step in time. Do not interchange these meanings when reading code or drawing the pipeline.



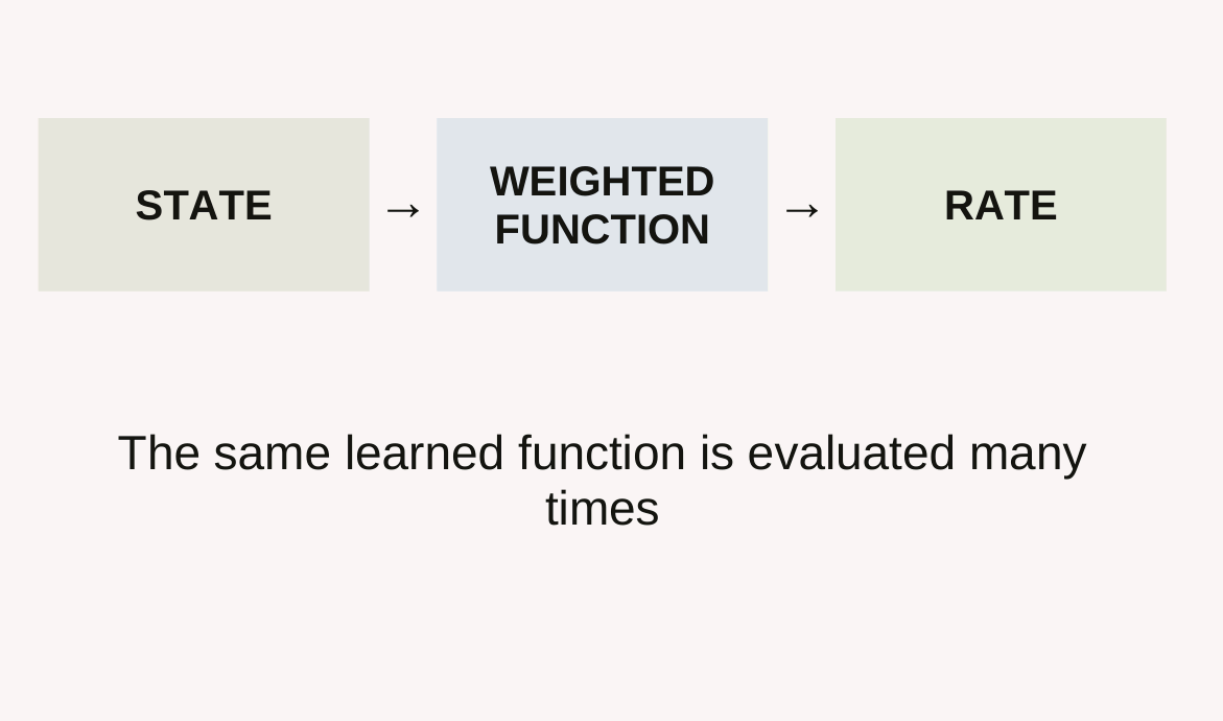
Sources: E1, N1. Full source details follow the glossary.

THE CORE

5. What does the neural network learn?

An adjustable function maps the current state to its rate.

The Arrhenius relation $k(T)=A\exp(-Ea/(RT))$ is a familiar nonlinear temperature-to-rate-constant map: for positive activation energy and fixed A, increasing temperature increases k. A reaction rate also depends on the mechanism and concentrations; k alone is not the entire state derivative. A neural rate model learns an adjustable map instead of identifying every weight with A or Ea. Function composition means passing one output into another function, such as state → normalization → rate model. A polynomial feature such as x^2 is one way to expose nonlinearity. The parameter vector θ collects neural network weights and biases. A vector field assigns a rate vector to every input state in its domain. The solver evaluates this field repeatedly, so a small rate error may accumulate during rollout. The adjustable-knob analogy refers here to network parameters; it does not mean that every weight is a chemical reaction rate constant. The small teaching model contains a deliberate decay prior. That prior is suitable for the chosen positive scalar decay, but would prevent modeling arbitrary species production and must not be transferred blindly to chemistry.



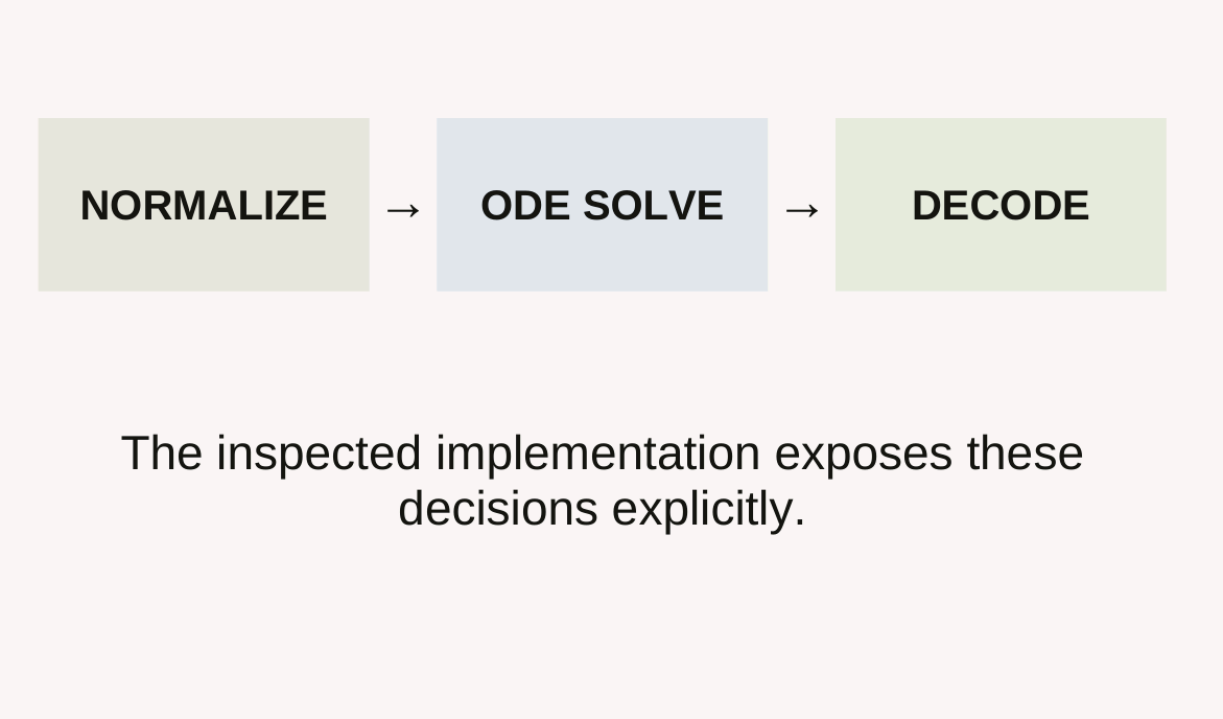
Sources: V1, E2, N3, H1. Full source details follow the glossary.

THE CORE

6. Where this appears in vZero

The inspected implementation exposes these decisions explicitly.

In the inspected source, model.py defines ChemistryODEFunc and NeuralODESurrogate. NeuralODESurrogate.forward normalizes the initial state, performs the configured solve, and either returns normalized output or decodes it according to the loss path. normalizers.py contains coordinate preprocessing; objectives.py contains alternative aggregation rules. This is a code inspection, not a fresh execution of every package preset. The documented package supports more than this single chemistry path, including classification and other research experiments. This guide focuses on the pieces needed to explain the handwritten Neural ODE ideas.

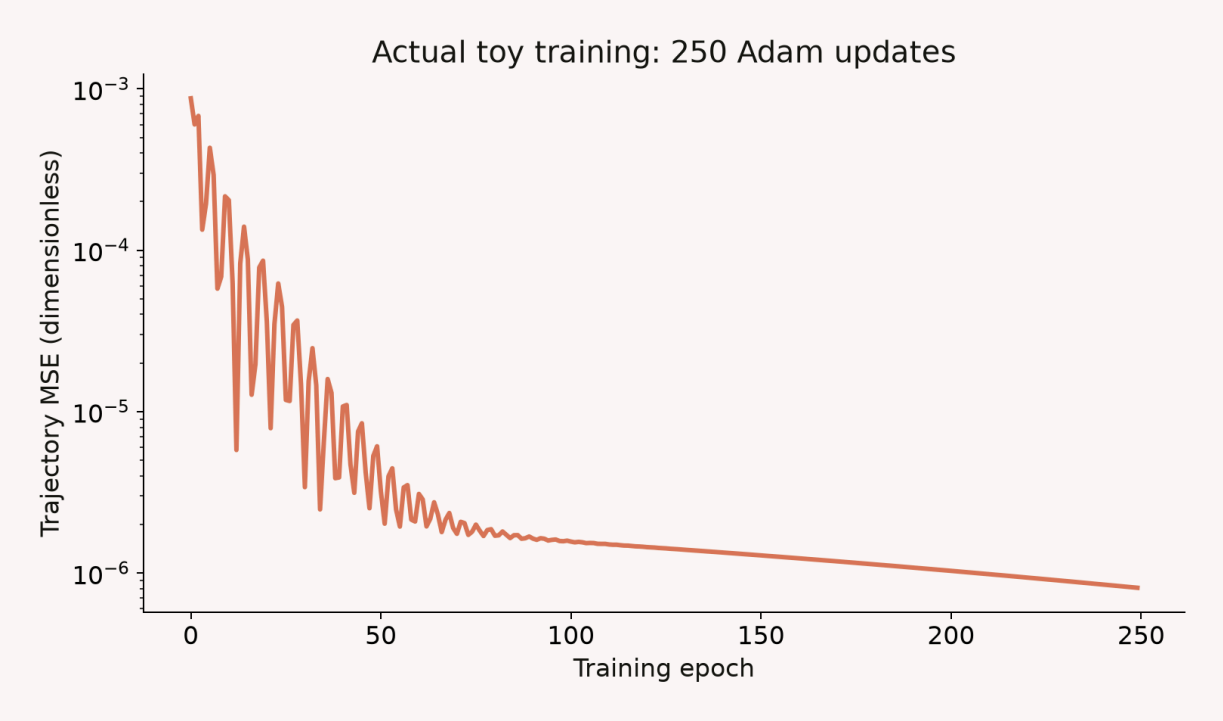


Sources: V1. Full source details follow the glossary.

7. A complete small learning loop

Generate a trajectory, measure its error, update the rate model.

The runnable experiment trains on initial values 0.5, 1.0 and 1.5 over dimensionless time 0 to 2. It uses a 1→12→1 tanh network inside a positive-decay rate expression and differentiates through fixed-step RK4. Seed 7 and float64 are recorded. The curve is the actual measured trajectory MSE before each update, not a fabricated illustration. This bounded run demonstrates the learning loop. It provides no evidence about the accuracy or speed of a trained H2/air model. Inspect data/training_metrics.json for executed values.




```
pred = odeint(
    rate, y0, t, method="rk4")
loss = ((pred - target)**2).mean()
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

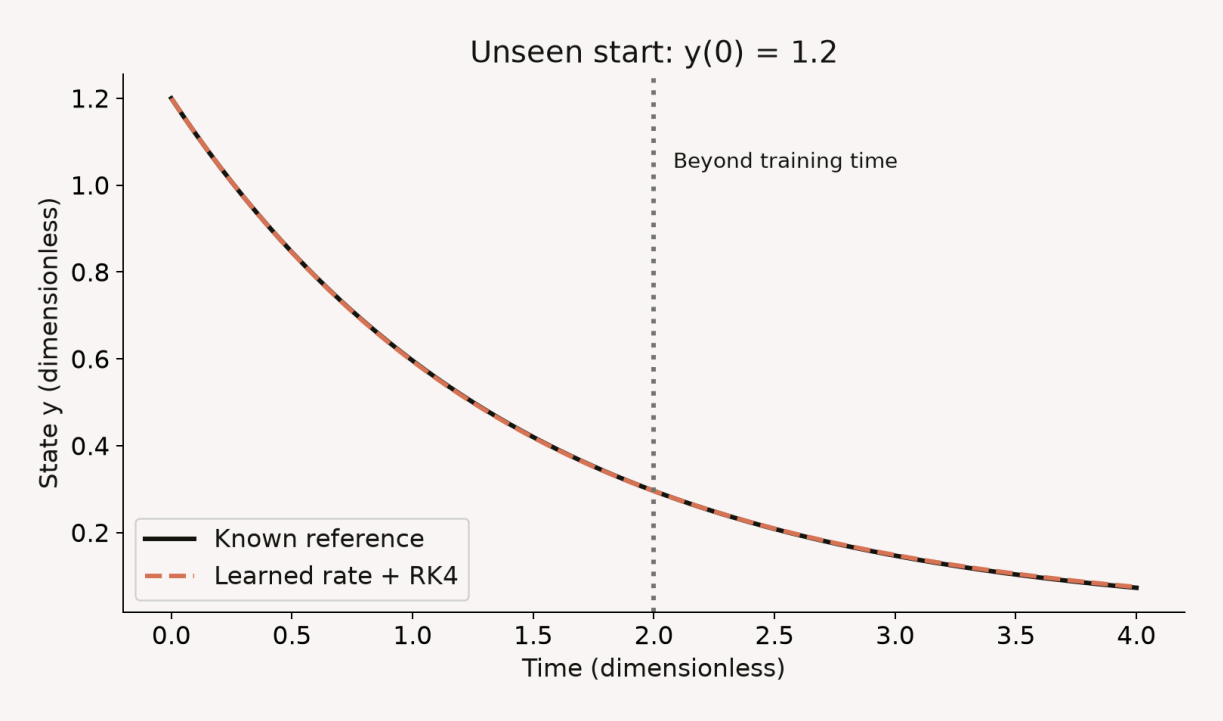
Sources: E2. Full source details follow the glossary.

THE CORE

8. Test a rollout beyond the training window

A good training curve is only the first check.

The test starts from 1.2, a new initial value inside the training initial-condition range. Times after 2 extend beyond the training horizon; this is temporal extrapolation on the toy, not extrapolation to a new reaction mechanism or thermodynamic regime. Compare every point with the analytic solution. The observed RMSE is about 0.00107 in dimensionless state units for this seeded run. That number belongs to this example only. More demanding tests change initial conditions, horizon, solver tolerance and physical conditions independently.



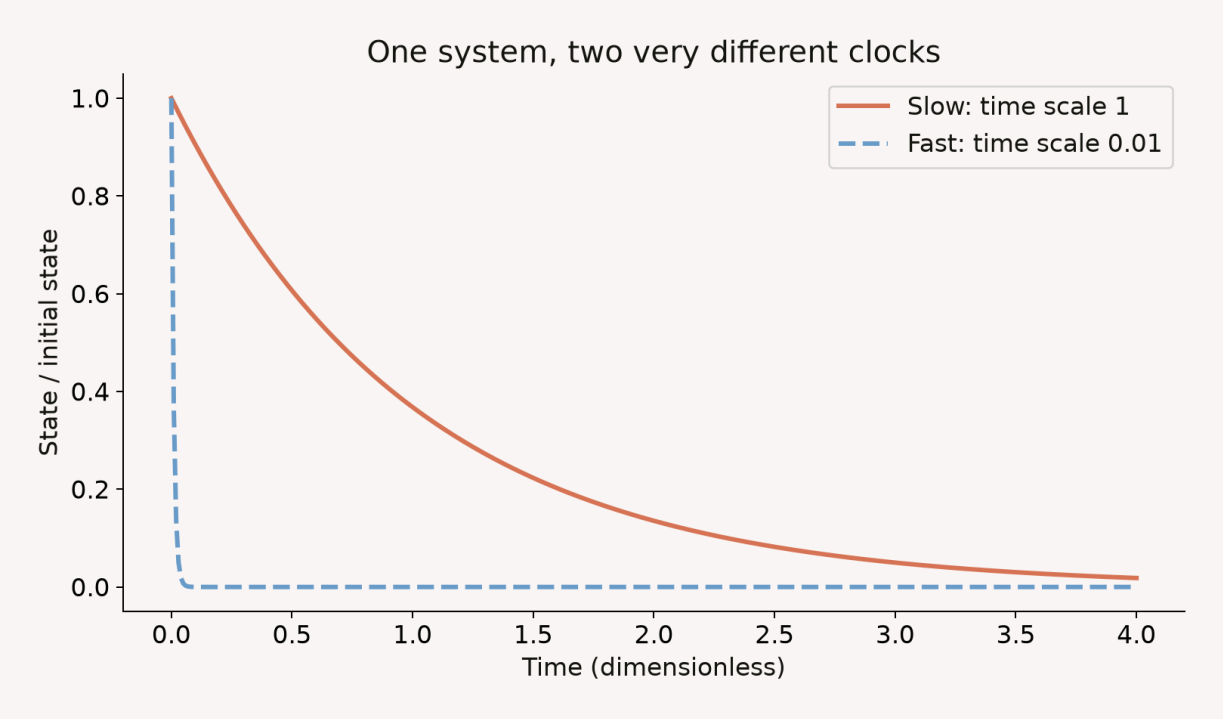
Sources: E2. Full source details follow the glossary.

NUMERICS

9. Stiffness means competing clocks

A very fast component can restrict steps over a slow process.

Our two independent decays have rates −1 and −100. The fast component dies away quickly, while the slow component still evolves. An explicit solver may remain constrained by numerical stability even when the fast component looks visually negligible. This is a simple real-eigenvalue illustration of stiffness; general nonlinear systems also depend on local behavior, coupling and the integration method. In chemistry, separate reaction timescales make solver choice an essential modeling decision. Stiff Neural ODE research addresses these difficulties, but a neural network does not automatically remove them.



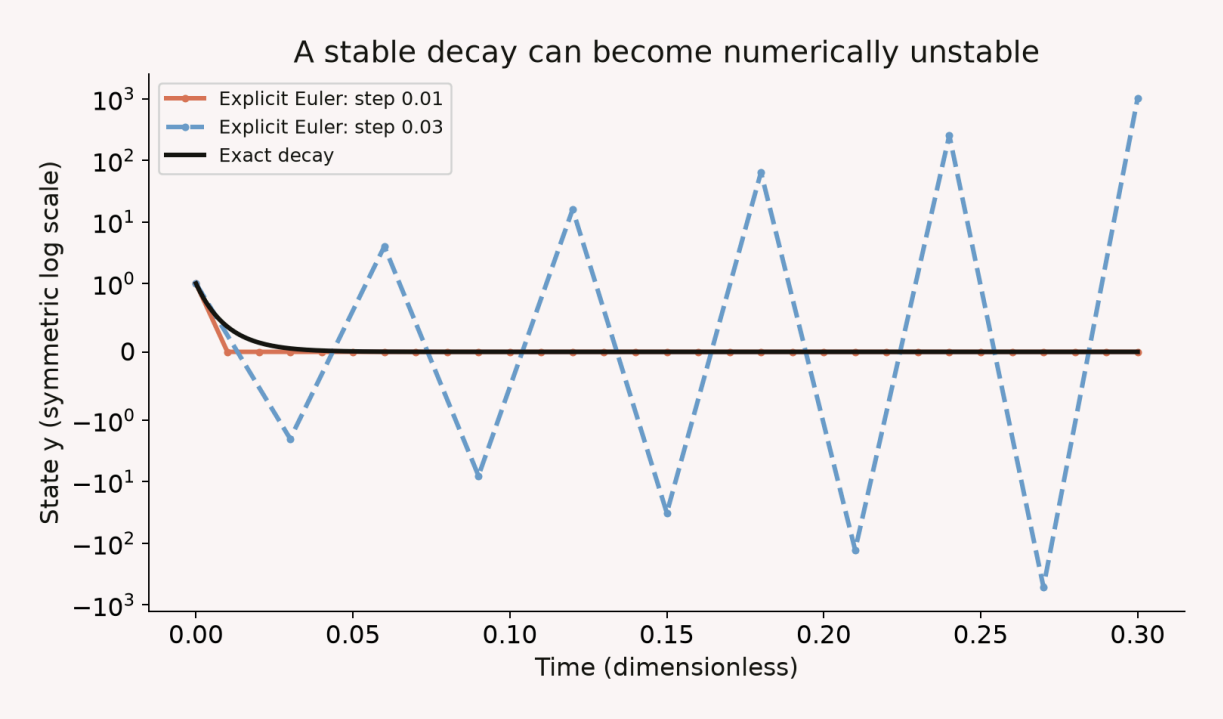
Sources: E1, N2. Full source details follow the glossary.

NUMERICS

10. The solver is part of the model you run

Stable physics can still produce unstable numerical predictions.

For $y' = -100y$, explicit Euler updates by $y_{next}=(1-100h)y$. At $h=0.03$, the multiplier is -2, so alternating values grow even though the exact solution decays. This calculation illustrates stability; it is not a test of vZero's RK4 or adaptive methods. For stiff systems, implicit methods such as Radau and BDF are standard options; their computational and differentiation requirements must also be considered. Solver success flags, tolerance changes and reference comparisons are needed before interpreting a learned trajectory. Numerical stability is distinct from training convergence.



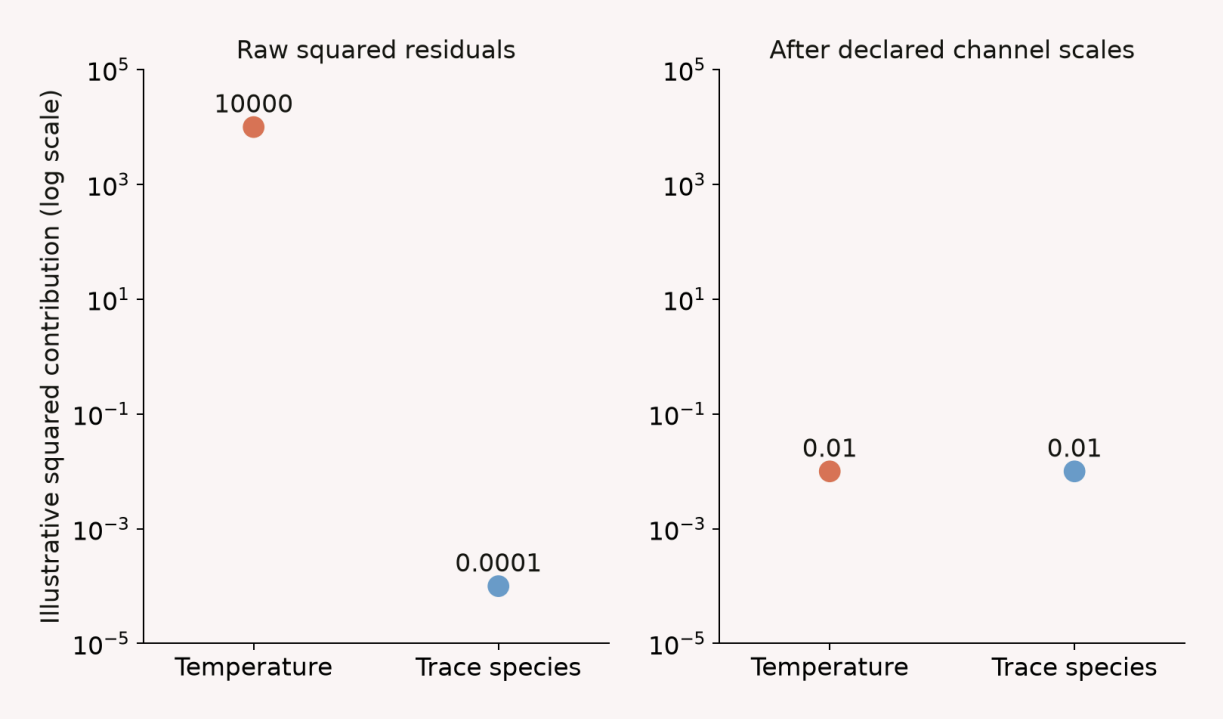
Sources: E1, S1. Full source details follow the glossary.

TRAINING

11. Normalize the microphones

Declared channel scales change which residuals the loss hears.

The synthetic comparison uses residuals 100 and 0.01, then declared scales 1000 and 0.1. Raw squares are 10000 and 0.0001; scaled squares are both 0.01. The example makes a units problem visible, not a claim that these channels deserve identical scientific importance. Dividing a residual by its scale is equivalent to weighting its square by the inverse square of that scale. Fit data-dependent scales using the training partition. Coordinate scaling can also change network behavior and solver tolerances, so the residual identity alone does not prove identical training outcomes.



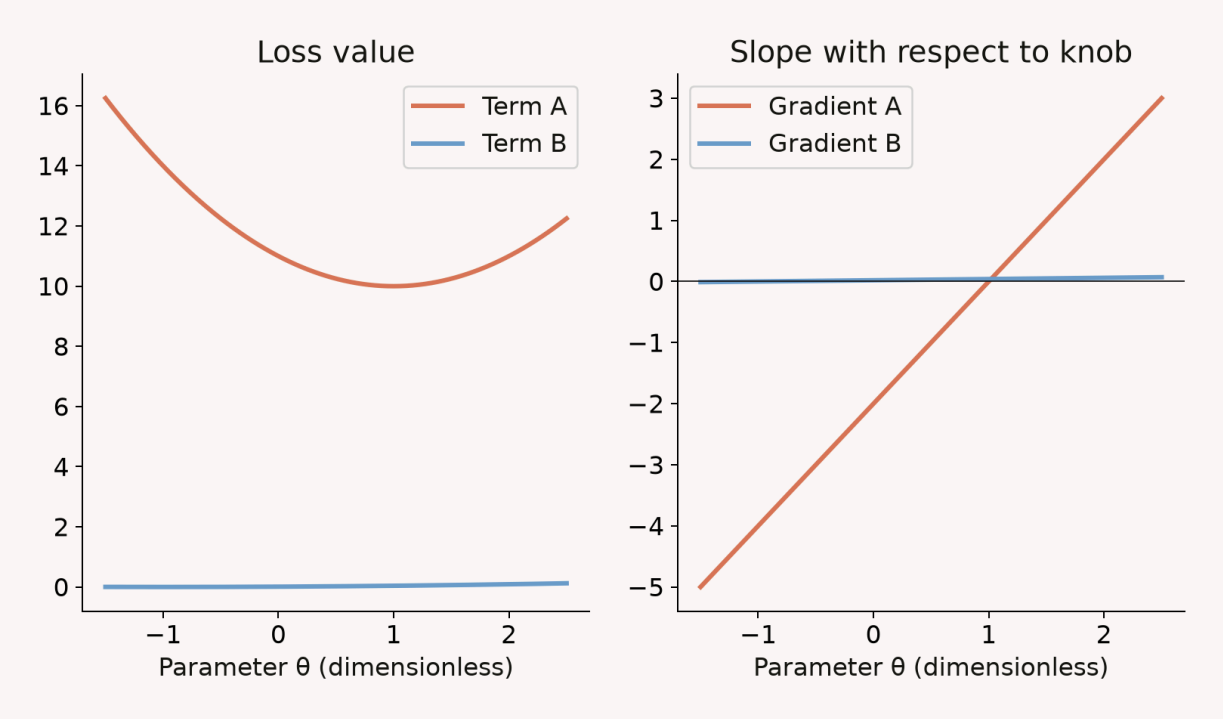
Sources: E1, V3. Full source details follow the glossary.

TRAINING

12. Loud loss is not loud gradient

Measure the push on the weights, not only the score value.

A gradient is a sensitivity of the loss to parameters. The demonstration adds a constant 10 to one quadratic term; the constant raises its loss value but leaves its derivative unchanged. The historical research notes corrected an earlier comparison that mixed different definitions of a logged fit column. They also report changing gradient balance during training. A ratio of logged losses is not a measured gradient share. For a Neural ODE, calculate each term’s parameter gradient under the actual aggregation, inspect its norm and alignment, and then evaluate whether a weighting change improves physical outputs.



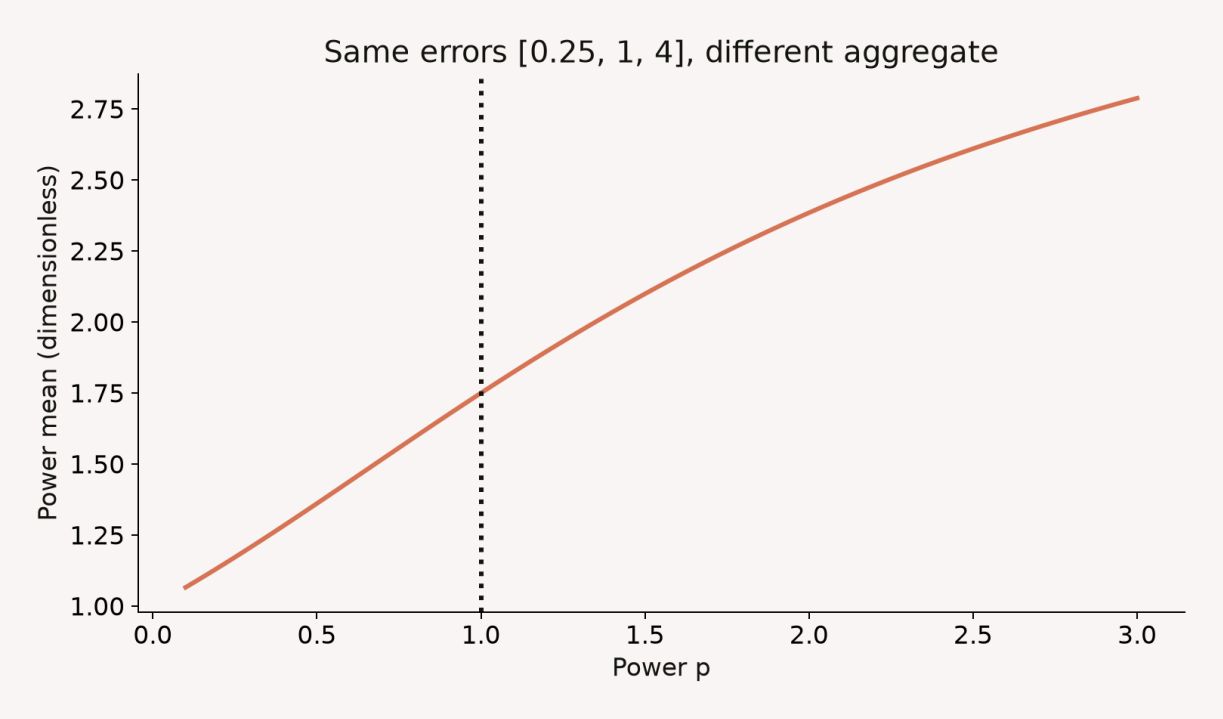
Sources: E1, V2. Full source details follow the glossary.

TRAINING

13. The power mean is an emphasis knob

It combines positive channel scores; it does not change chemistry.

Given positive normalized scores s, compute mean(s**p)**(1/p). The plotted example holds [0.25,1,4] fixed and changes p, so a lower plotted aggregate is not evidence that a model became more accurate. With p below 1, marginal emphasis shifts toward smaller input scores, for fixed positive weights and away from clipping boundaries. Reference calibration and p selection are separate choices. This training objective must not be confused with applying a fractional power to dynamical eigenvalues. Inspect vZero/objectives.py for the actual implementation, including numerical safeguards and reference handling.



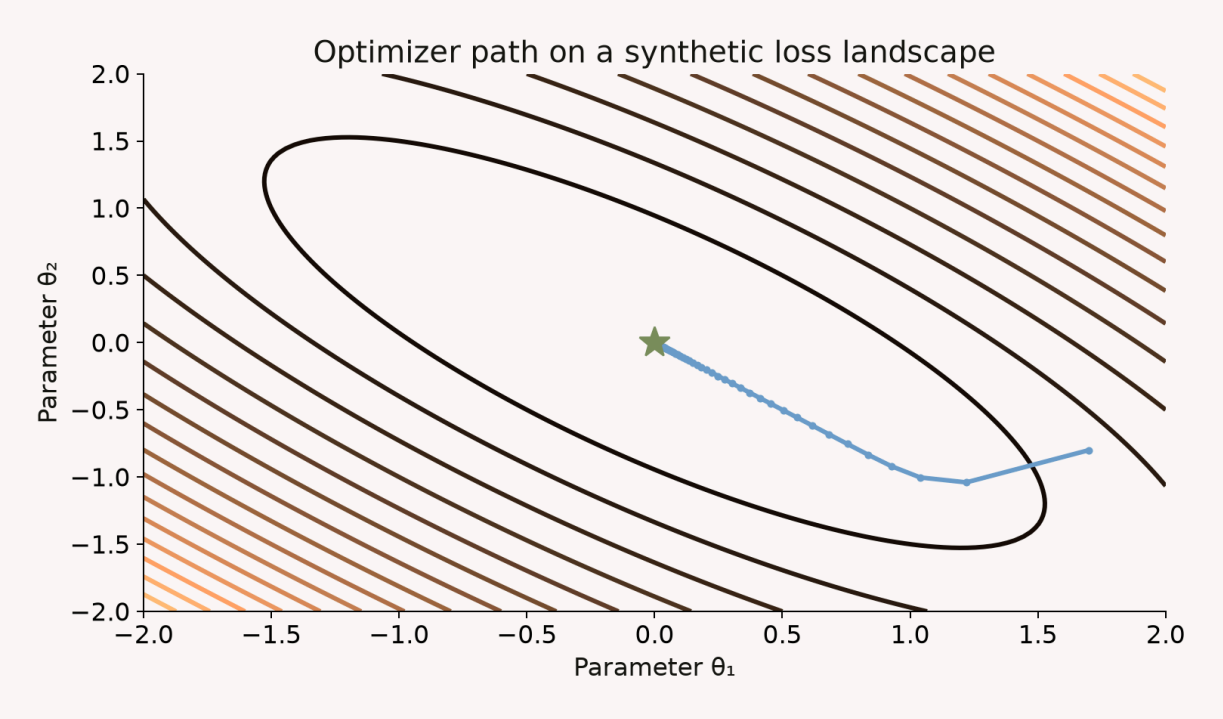
Sources: V1, E1. Full source details follow the glossary.

TRAINING

14. The landscape is over network weights

The optimizer moves the knobs; the solver moves the state.

This distinction prevents a common diagram error. A phase portrait plots physical or latent state and its time evolution. A loss landscape plots parameters and the score assigned to them. The displayed descent path was executed on a two-parameter quadratic; it is not a slice of vZero's trained loss and does not represent chemical kinetics. SGD, Adam and other optimizers update parameters between training iterations. The ODE solver updates the state within each prediction. vZero also contains a PPO-based weight-optimization experiment; its existence does not establish that reinforcement learning is the best training method.



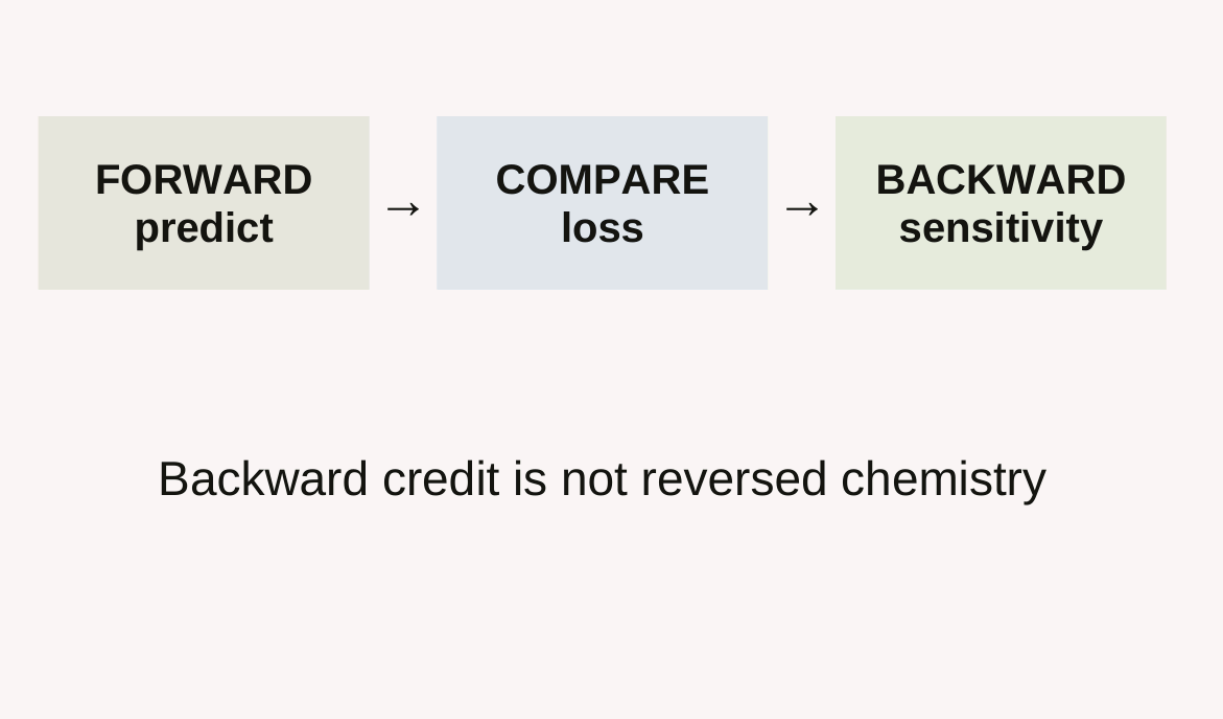
Sources: E1, V1. Full source details follow the glossary.

GRADIENTS

15. Forward prediction and backward credit

Training asks which earlier decisions caused the final mismatch.

In the forward pass, the solver follows the learned field from an initial state. After comparing predictions with measurements, training needs a derivative with respect to the network parameters. Lagrange multipliers are auxiliary variables that enforce a constraint in an optimization problem. Treating the ODE as a constraint and choosing time-dependent multipliers leads to the adjoint sensitivity equations. These multipliers are different from a mechanical Lagrangian, which is an energy function. Direct automatic differentiation follows the numerical operations actually executed. An adjoint is an auxiliary sensitivity problem that carries information backward; it is not a physical experiment running backward and does not imply reversed chemical reactions. If the loss includes observations at several times, their contributions must be accounted for at those times. These are ways to obtain gradients for an optimizer, not replacements for the optimizer itself.



Sources: N1, N4. Full source details follow the glossary.

16. Check the gradient on a problem you can solve

Direct and adjoint results should be tested against an independent answer.

At $\theta=-0.7$, $y(0)=1$ and $L=(y(1)-0.4)^2$, the exact derivative is $2(\exp(\theta)-0.4)\exp(\theta)$, about 0.095925685. The executed example compares this with a central finite difference, direct torchdiffeq differentiation and odeint_adjoint. Agreement here validates this simple calculation at stated tolerances. It is not proof that adjoints remain accurate on stiff chemical trajectories. Direct differentiation and a continuous adjoint can differ numerically; tight tolerances, checkpointing or alternative sensitivity approaches may be needed for difficult systems.

Analytic	**0.095925685**
Finite difference	**0.095925685**
Direct autograd	**0.095925685**
Adjoint	**0.095925685**

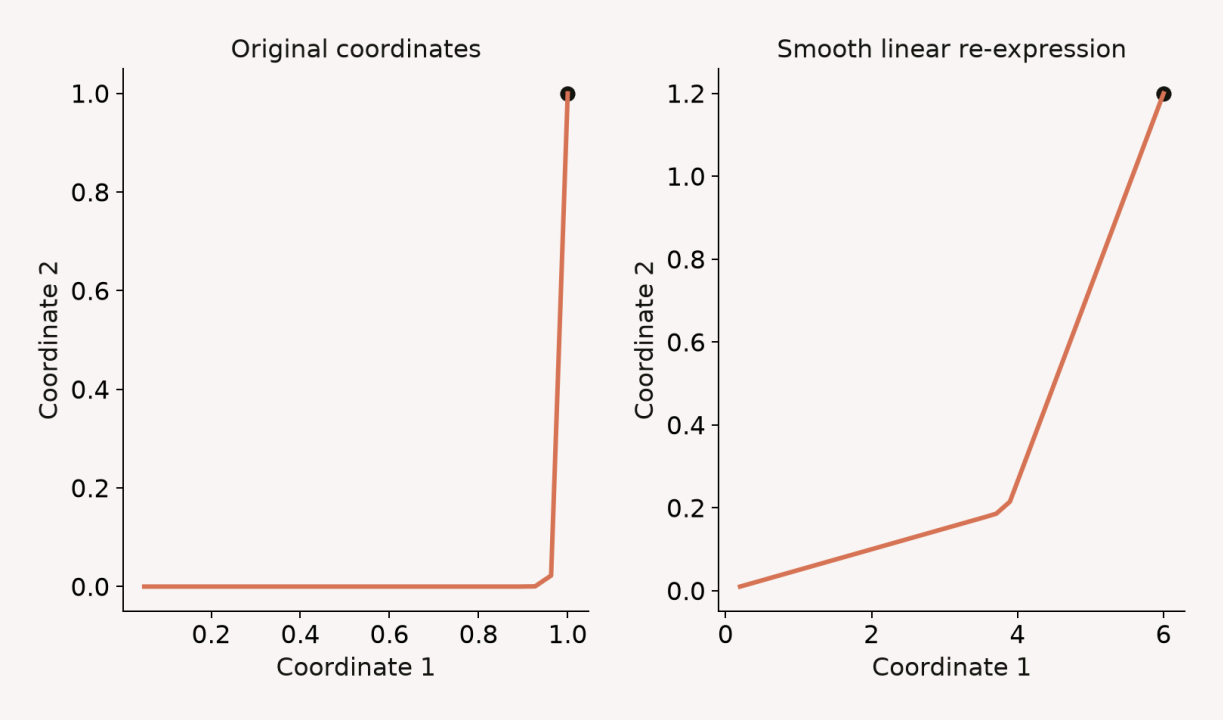
Sources: E2, N4. Full source details follow the glossary.

REPRESENTATION

17. Change the coordinates, keep track of the meaning

A coordinate map changes the description of a state.

A transformation can make patterns easier to represent. For $z=\Phi(y)$, the transformed rate is $z'=D\Phi(y)f(y)$ when the map is differentiable. If a network is trained directly to predict z', its decoded raw field is determined by the inverse map and its derivative. Therefore preprocessing, model coordinates and the loss space must be specified together. An arbitrary output transformation applied after training is not equivalent to retraining in new coordinates. The figure applies an invertible linear map to the same exact trajectories; the geometry changes while the underlying solutions correspond.



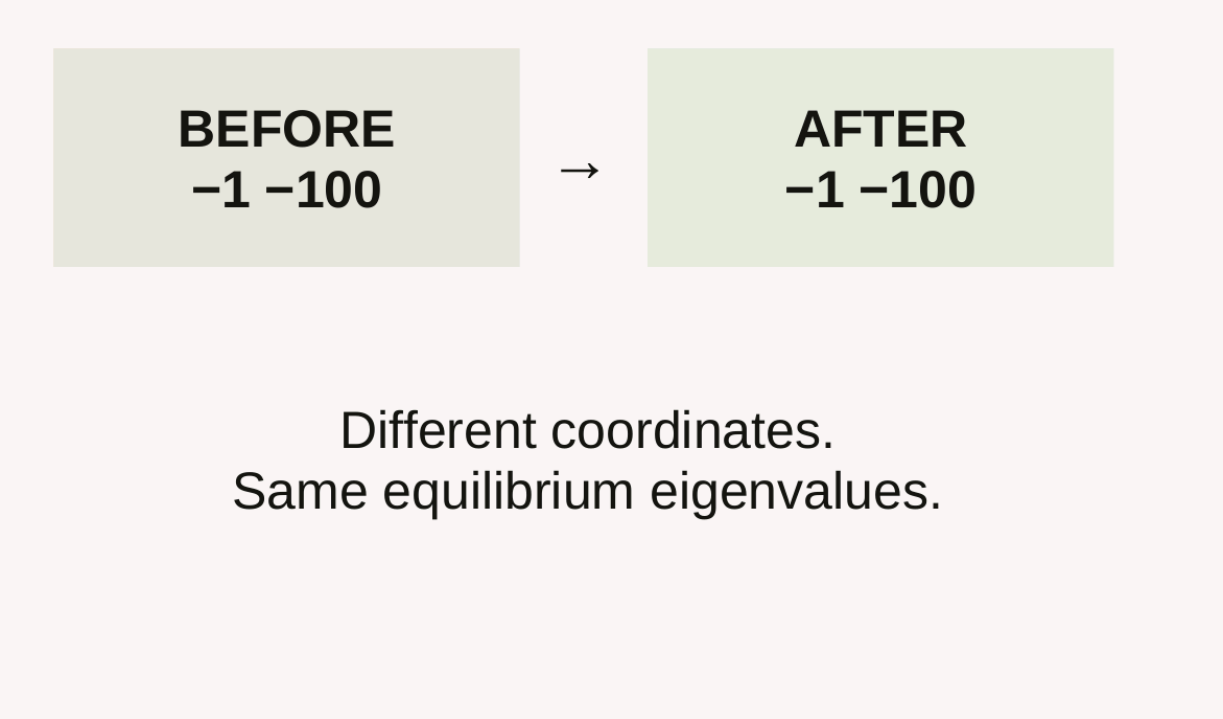
Sources: E1, V3. Full source details follow the glossary.

REPRESENTATION

18. A new map does not erase every fast clock

At equilibrium, smooth invertible coordinates preserve eigenvalues.

At an equilibrium of a differentiable vector field, a smooth locally invertible coordinate change produces a similarity transform of its Jacobian. Similar matrices have the same eigenvalues. The executed 2×2 example verifies this numerically. Away from equilibrium, nonlinear coordinate Jacobians contain additional terms, and practical conditioning can change. Singular maps, time reparameterization and deliberately changed dynamics require separate analysis. Thus log or power transformations may help learning or numerical behavior, but should not be advertised as a general proof of eigenvalue compression for an equivalent chemical system.



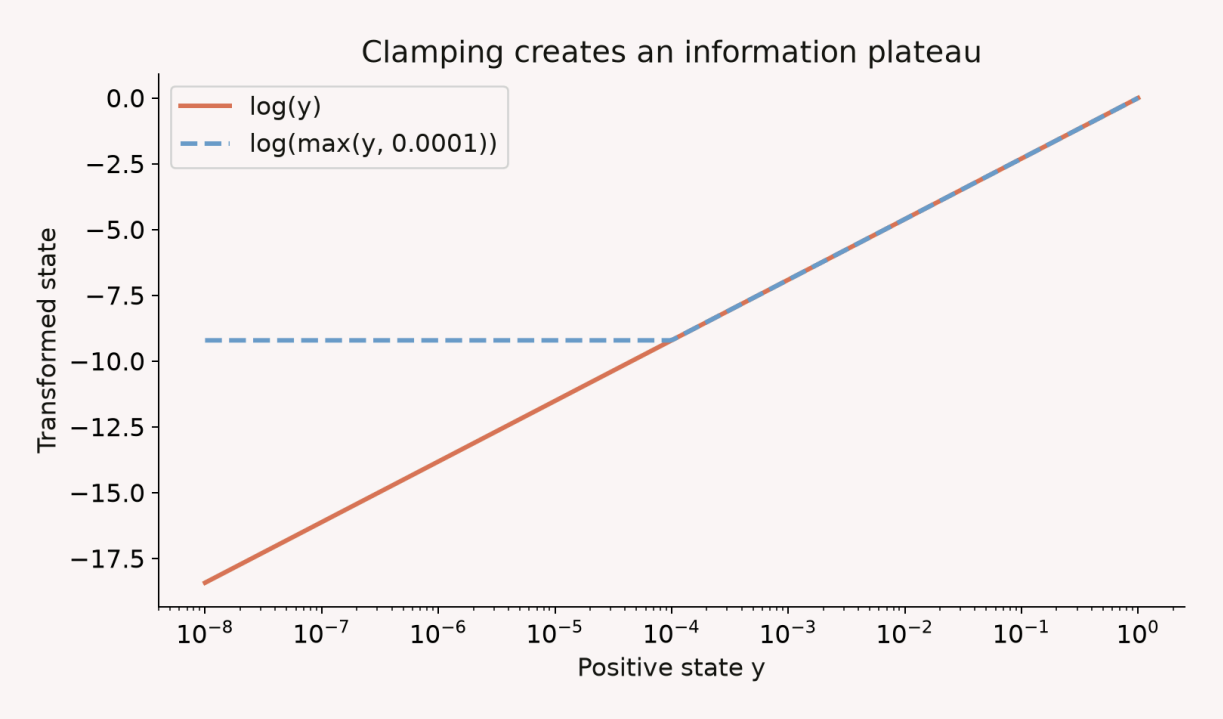
Sources: E1, V3. Full source details follow the glossary.

REPRESENTATION

19. Log transforms need a boundary policy

Small concentrations and exact zeros are different cases.

A pure logarithm is undefined at zero. Replacing y with max(y,epsilon) creates a plateau: multiple raw values share one encoded value and the derivative through the clamp vanishes below the threshold. Adding epsilon is a different transformation with a different decoded vector field. For z=log(y), a learned finite transformed rate gives y′=y z′ only on the positive domain; extending that expression to zero requires an explicit boundary argument. This matters in chemistry because a species absent initially can later be produced. Test round trips, zeros, negative numerical excursions and production from zero.

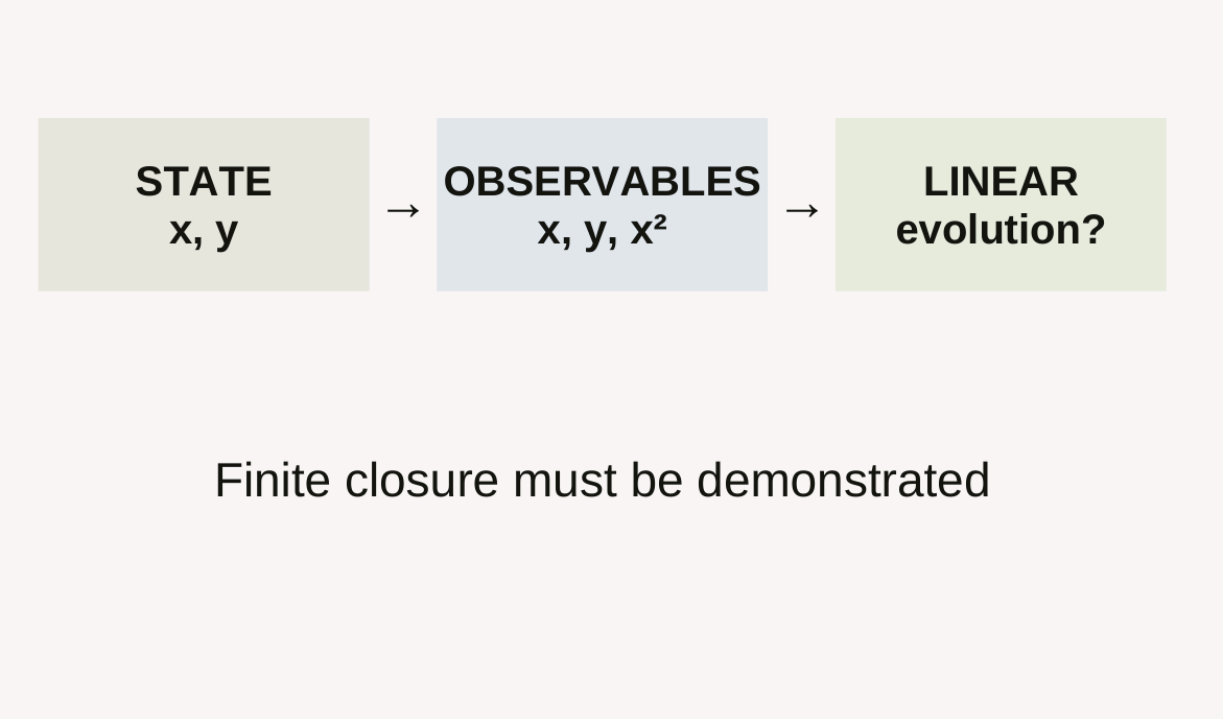


Sources: E1, V3. Full source details follow the glossary.

20. Koopman changes what we observe

Nonlinear state dynamics can act linearly on functions of state.

The word operator here describes how measurement functions evolve under a dynamical flow. It is linear in the observable function, even when the original state evolution is nonlinear. Exact Koopman descriptions generally require an infinite-dimensional function space. Finding a useful finite collection that evolves approximately linearly is a modeling task, not a consequence of adding polynomial features alone. This is the connection to the handwritten composite-function idea. A Koopman-inspired representation could be studied within a vZero experiment; these notes do not establish that the inspected chemistry workflow already uses one.

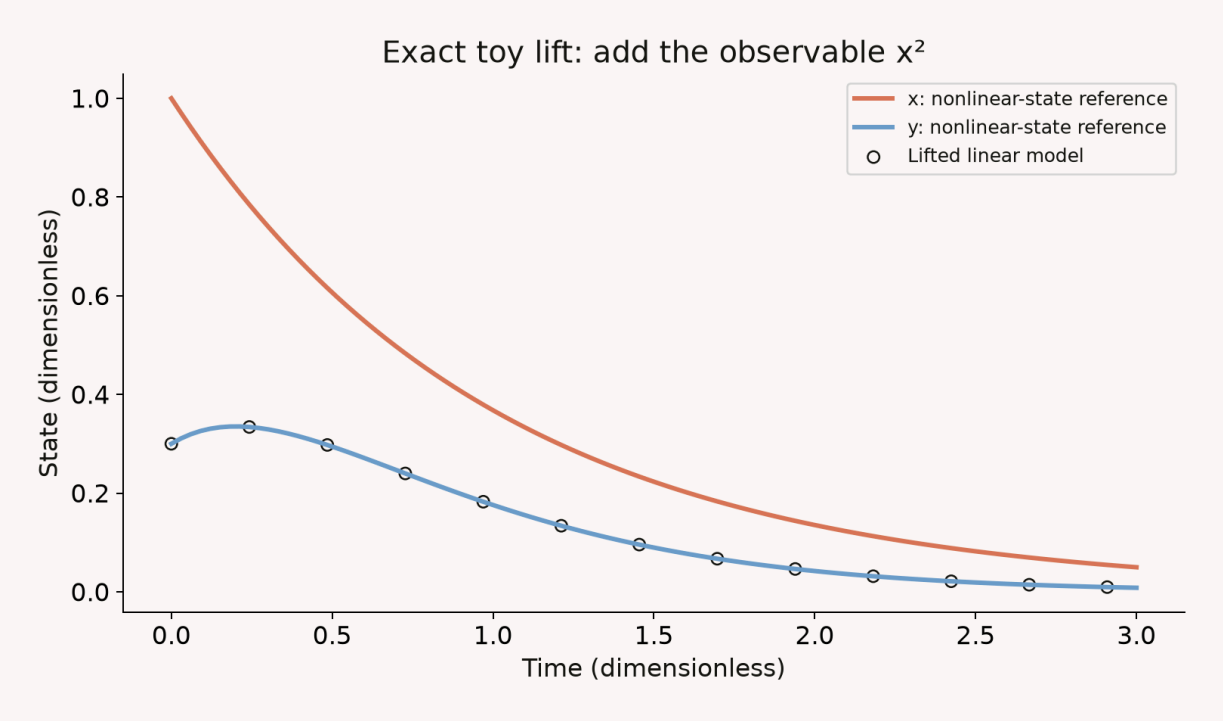


Sources: K1, K2, H1. Full source details follow the glossary.

21. One extra observable closes this toy

Add x^2 and a nonlinear system becomes a three-variable linear system.

The lifted vector [x,y,q] obeys a constant matrix equation with rows [−1,0,0], [0,−2,1], [0,0,−2]. It is exact for initial $q=x^2$. We independently compare the matrix exponential with $x(t)=\exp(-t)$ and $y(t)=\exp(-2t)(0.3+t)$. Closure means that differentiating every chosen observable produces only combinations already in the chosen set. A different nonlinear term could require x^3, then x^4, and so on. This example shows the attraction and the key limitation of lifting without claiming closure for a combustion mechanism.



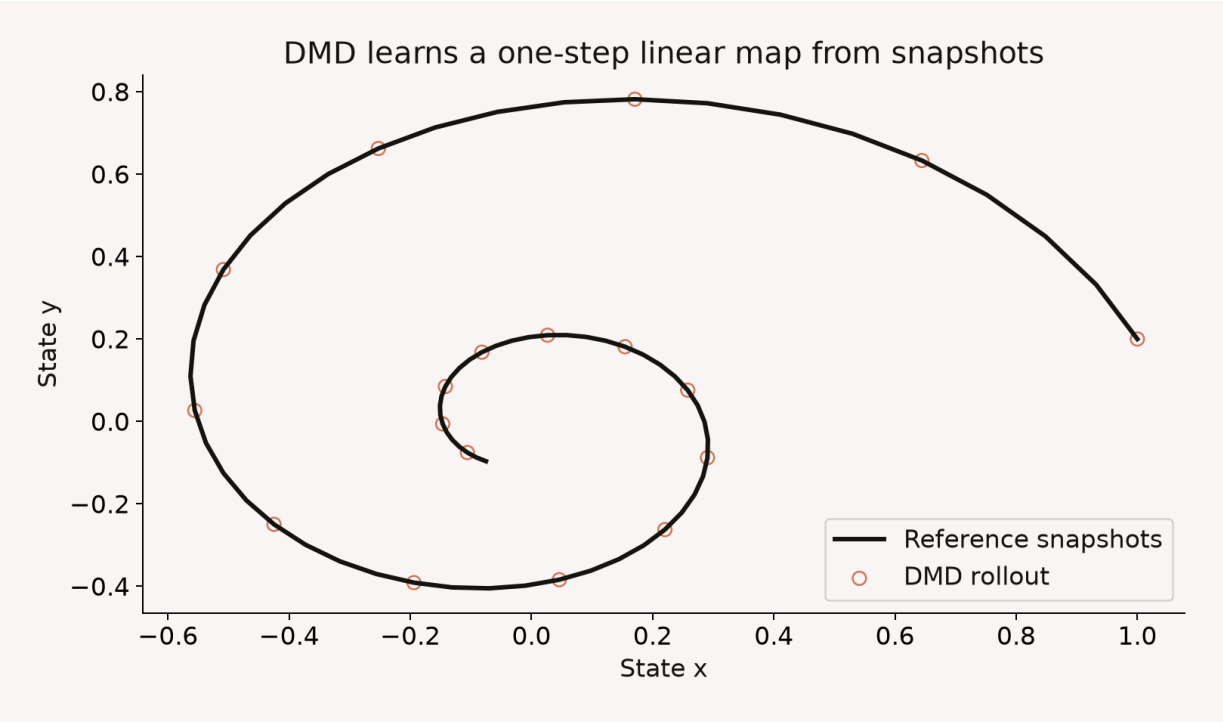

```
K = [[-1, 0, 0],
     [ 0,-2, 1],
     [ 0, 0,-2]]
lifted_t = expm(K*t) @ [1, .3, 1]
```

Sources: E1, K2. Full source details follow the glossary.

22. DMD fits a linear step from snapshots

Pair each measured snapshot with the one that follows it.

Dynamic mode decomposition estimates linear evolution from snapshot data. For the turbulence connection in the notes, a snapshot can be a long vector containing velocity measurements across a spatial grid at one instant. Repeating measurements supplies columns of a data matrix; a dominant spatial pattern is a mode. The same organization can be used for temperature or concentration fields, with different physics and measurement limitations. The teaching example uses a known damped rotation and computes A=Y pinv(X). Its reconstructed rollout matches the noiseless generating map to numerical precision. In nonlinear applications, a fitted linear map is usually an approximation over the sampled region; selected observables can improve its usefulness. Sampling interval, measurement noise, data rank and out-of-sample behavior matter. DMD is not synonymous with a general Neural ODE: it fits a discrete map, while a Neural ODE parameterizes a continuous-time rate.



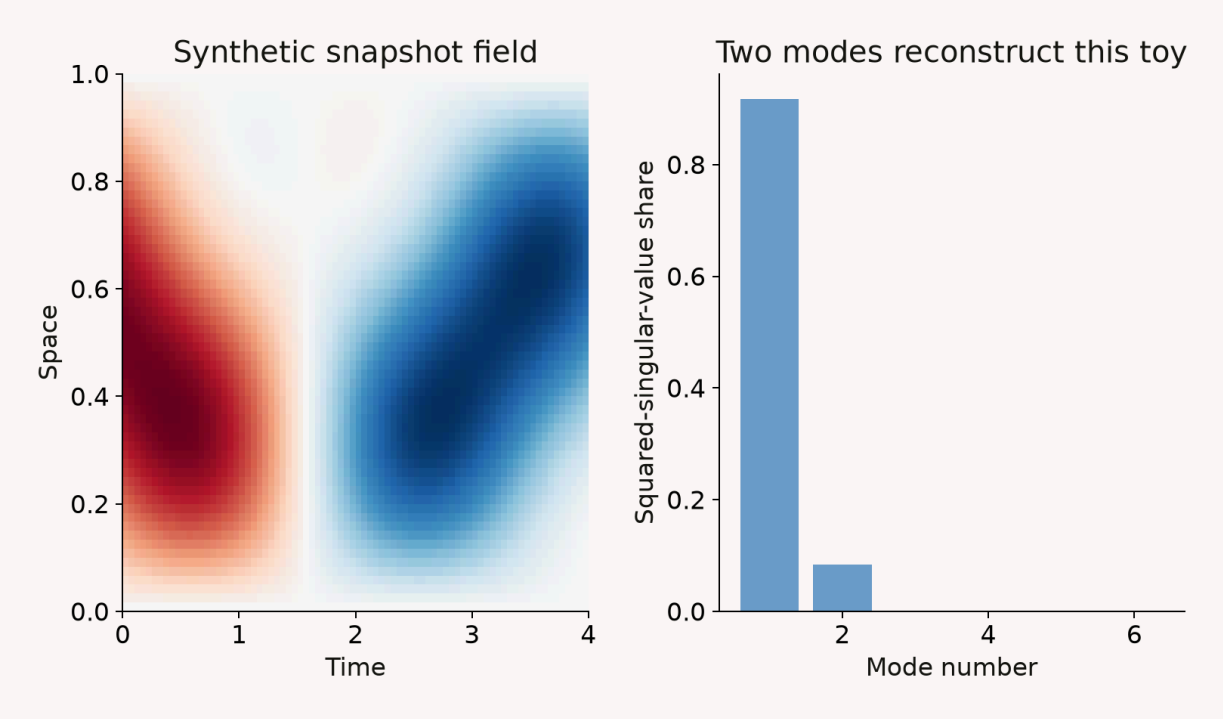

```
X = snapshots[:, :-1]
Y = snapshots[:, 1:]
A = Y @ np.linalg.pinv(X)
next_state = A @ current_state
```

Sources: E1, D1. Full source details follow the glossary.

23. POD compresses snapshots

Compact reconstruction does not yet give a predictive dynamics model.

Proper orthogonal decomposition can be computed using the singular value decomposition of a snapshot matrix. Our synthetic field is built from exactly two spatial patterns, so two modes reconstruct it up to numerical precision. This construction explains the method; real data need not have rank two. A small reconstruction error does not tell us how the retained coefficients will evolve in the future. Trace species or short events may contribute little variance but matter physically. The historical vZero research notes also warn that correlation between species does not automatically make a joint low-rank basis superior.

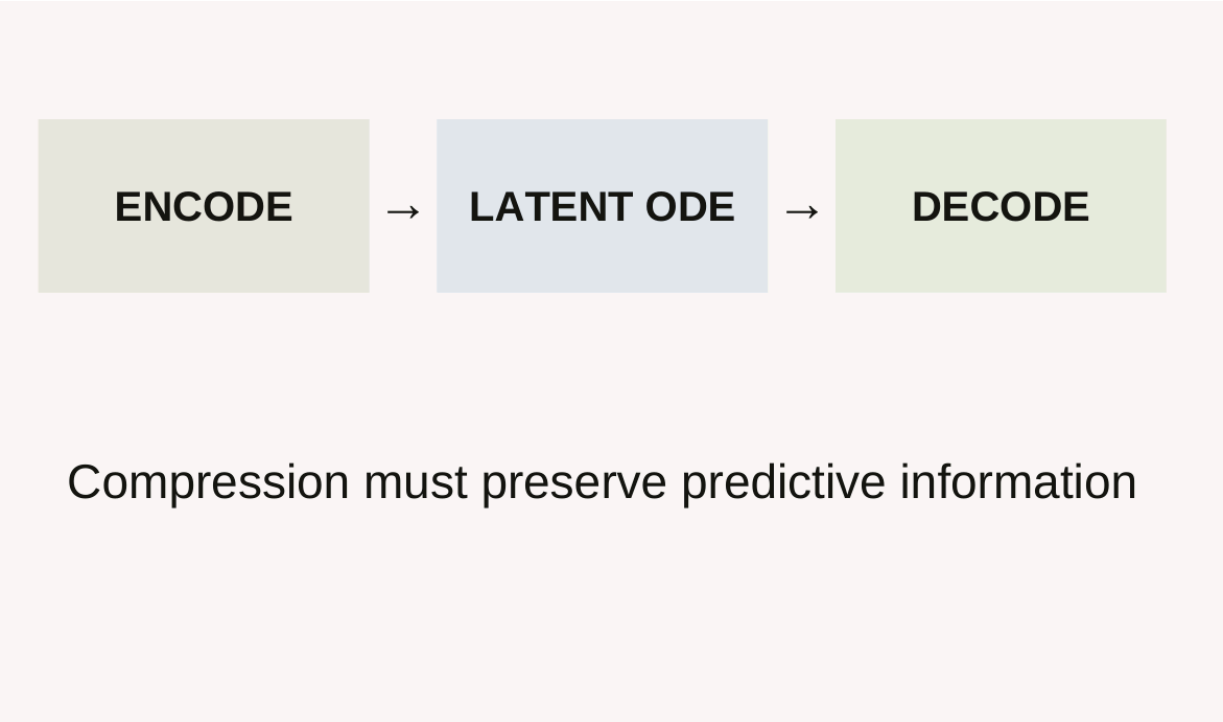


Sources: E1, V2, H1. Full source details follow the glossary.

24. A latent Neural ODE adds a learned evolution

Compress → evolve a hidden state → reconstruct observations.

A latent representation is an internal state learned or selected for modeling. Its dimension can be smaller than the observation vector, but information lost by compression may be necessary to predict the future. A useful test is whether similar latent states can have incompatible subsequent evolution. If so, the chosen latent variables are not a closed state description; more state, memory or inputs may be needed. Latent ODE models are an established research direction. Applying them to this project requires testing reconstruction, long-horizon dynamics and physical quantities together.



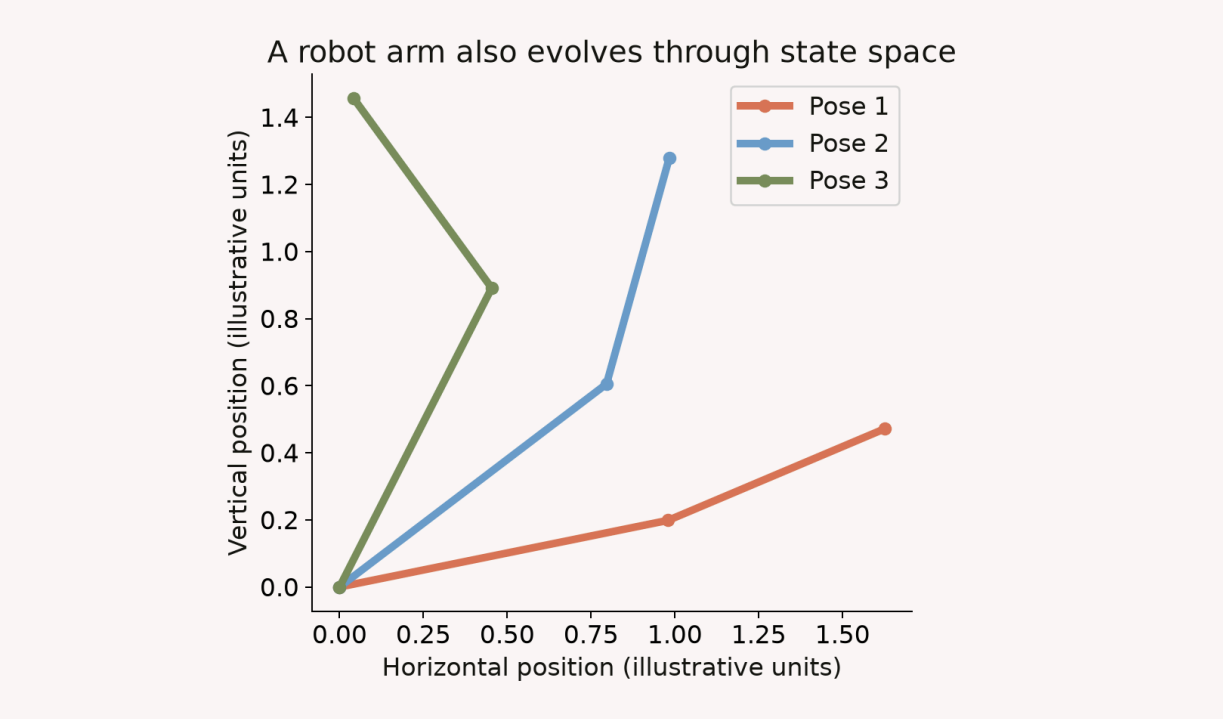
Sources: L1, E1. Full source details follow the glossary.

PHYSICS AND MEASUREMENTS

25. A robot arm makes state and rate tangible

Angles and velocities evolve; torques drive the motion.

For a mechanical system, write the state as joint angles q and angular velocities v. Its derivative contains v and the acceleration produced by torques, gravity and other forces. This makes the Neural ODE pattern concrete: the model evaluates a rate and the solver integrates it. The plotted poses are illustrative geometry rather than a trained robot model or experimental measurement. The connection is structural: both mechanics and chemistry have states, rates and constraints. The governing constraints themselves differ, so a robotics architecture is a useful comparison rather than an automatic chemistry solution.

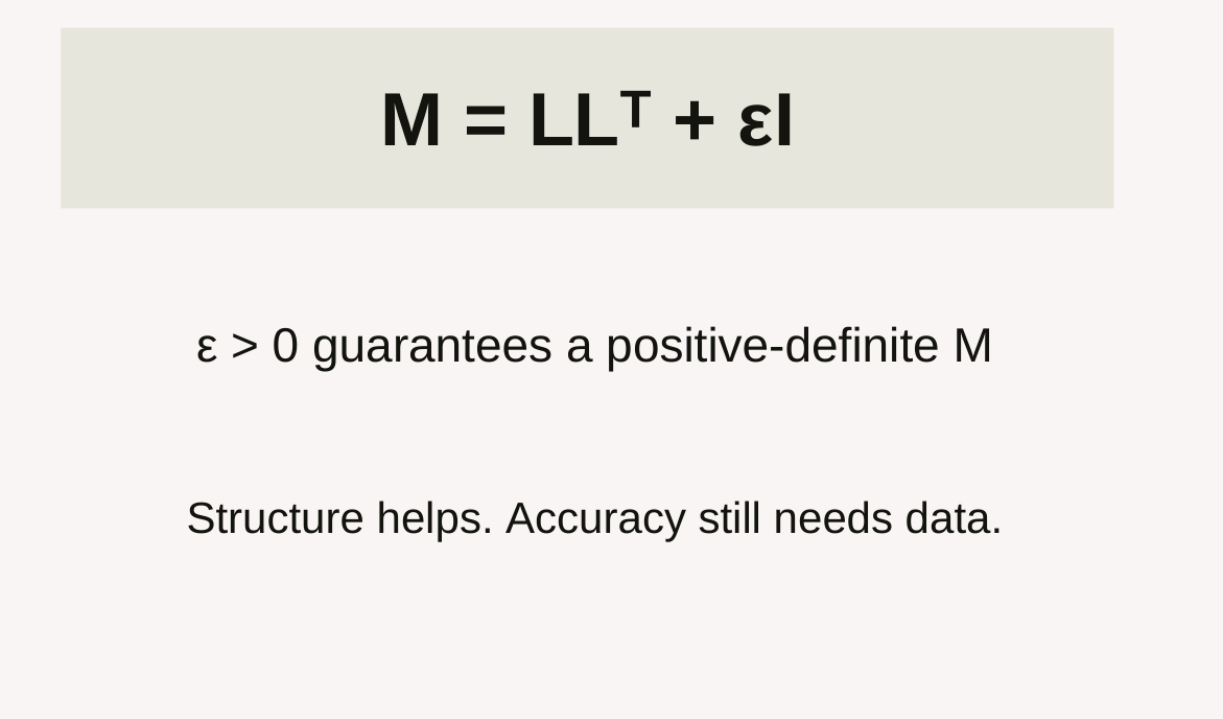


Sources: H1, R1. Full source details follow the glossary.

26. Build a valid mechanical mass matrix

Cholesky factors are structural knobs with a useful guarantee.

For any nonzero vector v, $v^T(LL^T+\text{epsilon } I)v$ equals the squared norm of L^Tv plus epsilon times the squared norm of v, which is positive. This gives an elementary way to constrain a learned mass matrix. A mechanical Lagrangian is usually kinetic energy minus potential energy. Equations derived from this energy description determine acceleration once forces and constraints are specified. Deep Lagrangian models use physical structure to organize mechanical dynamics; related Lagrangian Neural Networks learn an energy-based description. Friction and forcing need their own treatment. The Cholesky idea in the handwritten note belongs here. It is not a generic replacement for chemical species conservation or a guarantee of an accurate learned force law.

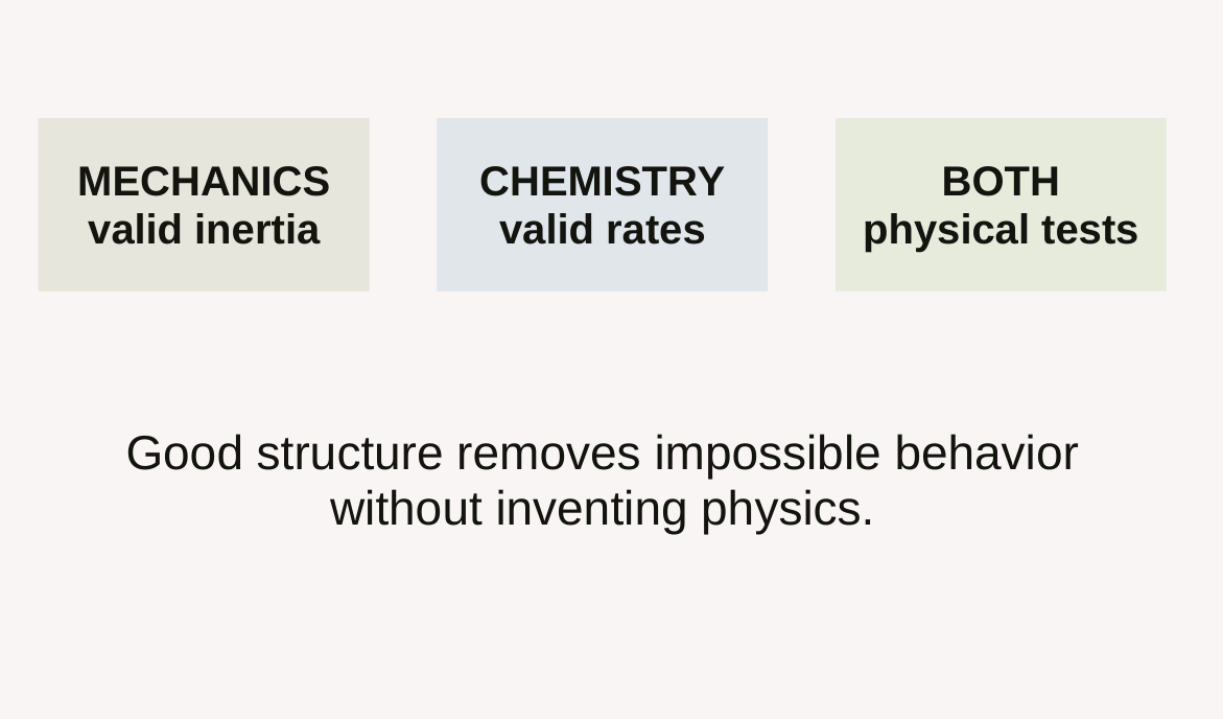


Sources: E1, R1, R2. Full source details follow the glossary.

27. Use constraints that match the domain

Good structure removes impossible behavior without inventing physics.

A reaction model can use a stoichiometric structure so that its rate lies in a conservation-compatible subspace. Positivity and energy balance are additional requirements, not automatic consequences of an arbitrary neural output. In an adiabatic reactor, temperature may rise while the appropriate total thermodynamic balance is respected; conserved total energy does not mean constant temperature. A mass-matrix factorization guarantees a mechanical matrix property and does not by itself enforce chemical invariants. For the selected nine-channel vZero example, do not enforce a unit sum on eight listed species without accounting for the omitted inert or other components.

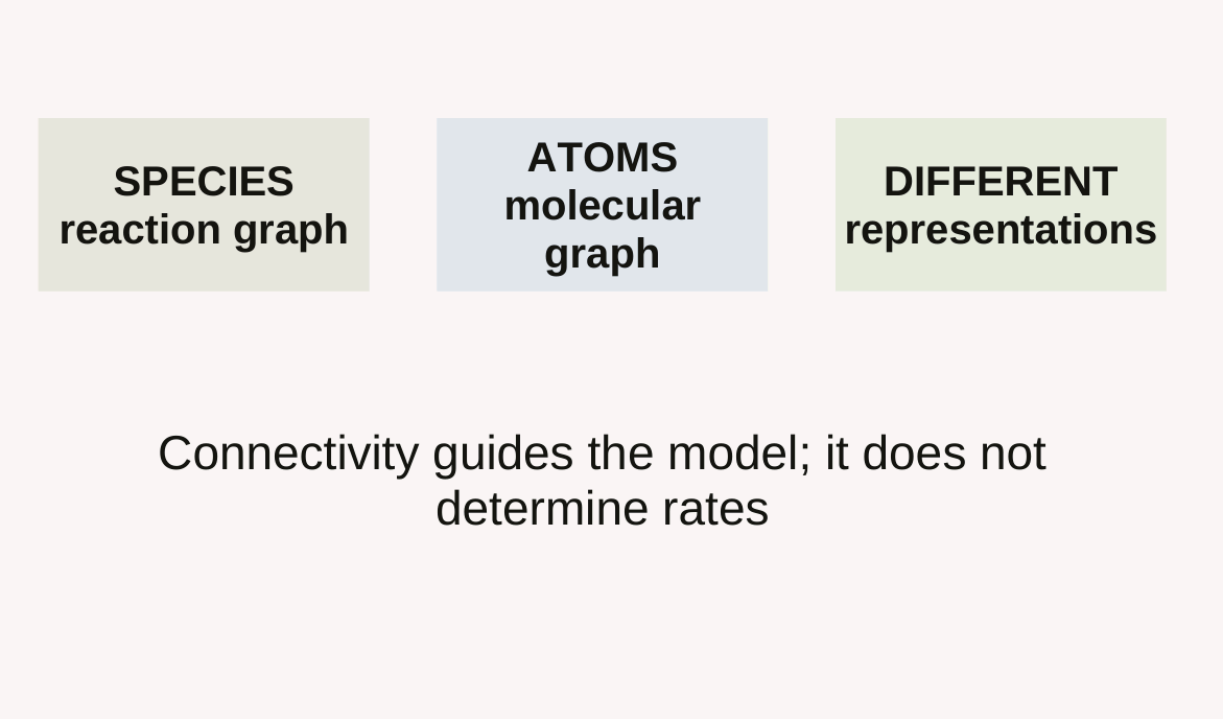


Sources: V1, V3, R1. Full source details follow the glossary.

28. A graph tells the model who interacts

Connectivity is prior information; it is not a complete rate law.

The inspected GraphConvFront in model.py combines a state's own features with mechanism-neighbor features derived through graph.py. This is evidence of a graph-aware front layer, not evidence that it improves every task. A reaction/species graph and an atom/bond molecular graph have different nodes and meanings. The handwritten graph-isomorphism and molecular-descriptor ideas relate to representation and symmetry. Isomorphism concerns equivalent graph structure under relabeling; it is not a numerical solver. Graph features can guide a learned rate, while physical constraints and rollout tests still need to be checked separately.

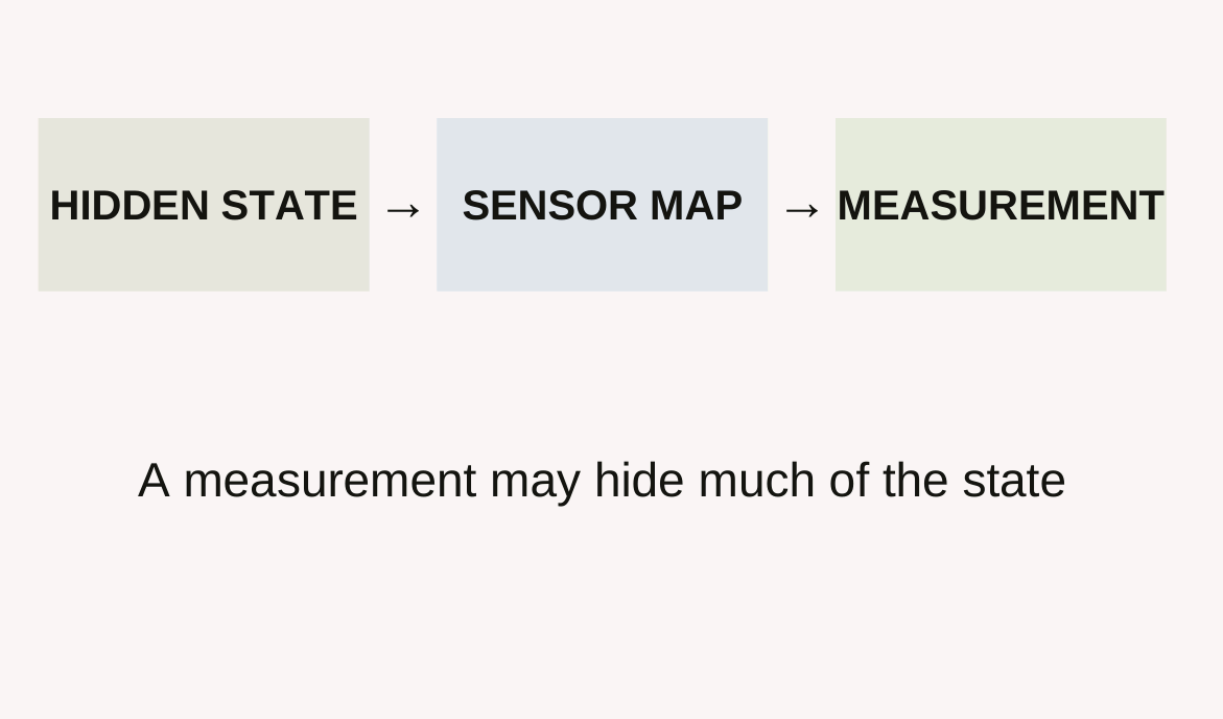


Sources: V1, H1. Full source details follow the glossary.

29. A camera sees an observation of the state

Write the measurement process before fitting its output.

Use observation = h(state) + noise to distinguish internal dynamics from measured signals. Chemiluminescence tomography uses projections to infer an emitting field under optical assumptions. Stereo digital image correlation instead estimates deformation/displacement from images; the handwritten acronym is read tentatively as stereo-DIC and needs user confirmation. Neither method is itself a Neural ODE. Their useful connection is an observation model or a source of training data. A flame image is not automatically a full chemical-state measurement, and camera brightness is not automatically temperature or heat release without an appropriate model.



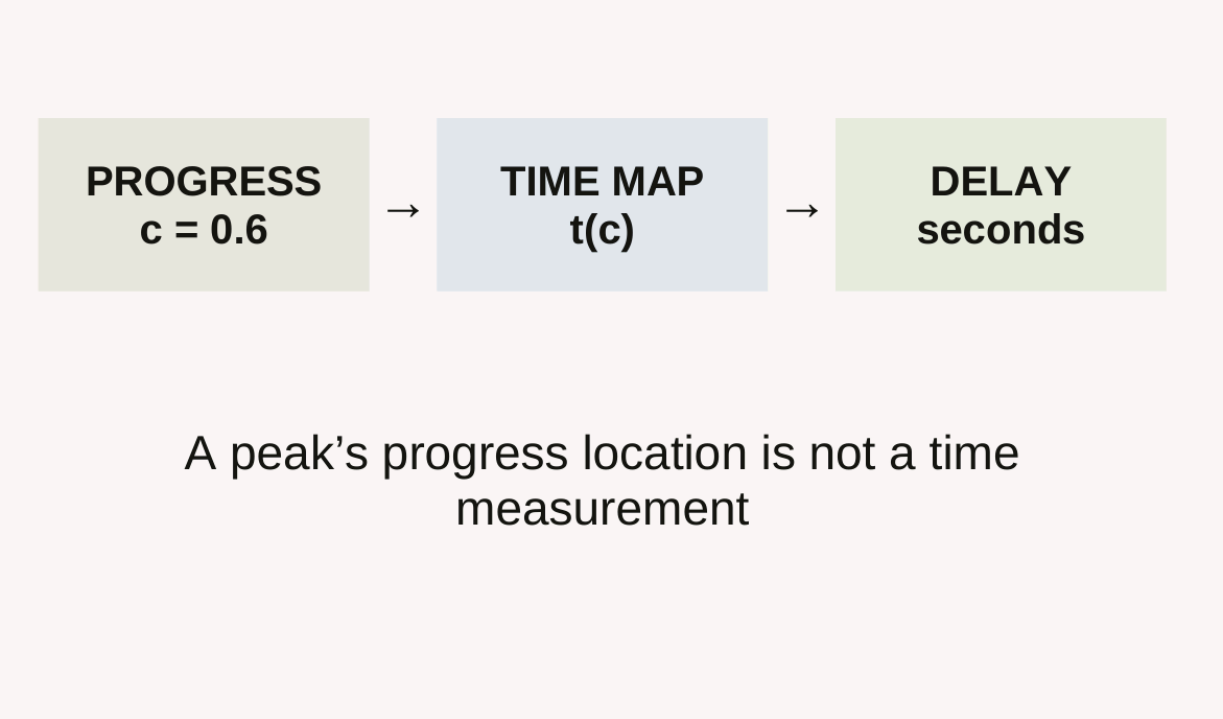
Sources: M1, M2, H1. Full source details follow the glossary.

VALIDATION

30. A progress coordinate is not a clock

Matching a curve’s shape can still miss ignition timing.

The research notes describe a progress variable c=(T−T0)/(T_eq−T0). A peak at c=0.6 gives a position along progress, not an ignition delay in seconds. Recovering time requires a valid monotone relation and a rate such as dc/dt, with special care where that rate vanishes or progress is not monotone. This distinction is central for chemical surrogates: matching species against progress may be valuable but does not establish correct time dynamics. Label each plot’s independent variable explicitly and define the ignition marker before reporting a timing error.



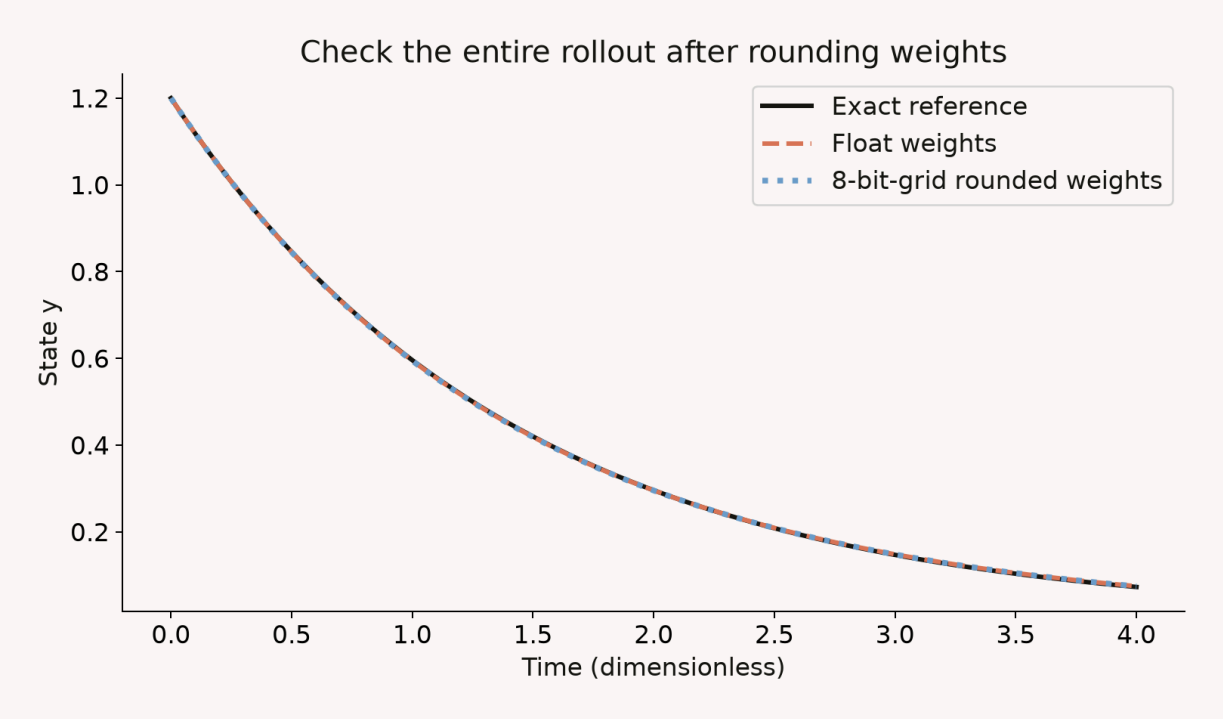
Sources: V2. Full source details follow the glossary.

DEPLOYMENT

32. Quantization is controlled rounding

Check rollout error after reducing numerical precision.

The toy rounds each weight tensor with a symmetric 8-bit grid and then performs the solve in float64. It measures sensitivity to weight rounding, not integer-kernel speed or memory savings in deployment. Errors introduced into a rate model can accumulate along a trajectory. Actual static INT8 quantization also needs representative calibration data; dynamic quantization handles some scales at runtime. Separately, the inspected vZero/quantize.py compresses visualization grids into uint8 values for browser transfer. That code is not evidence of INT8 inference for the network. Measure complete solver-plus-network performance on the target hardware.



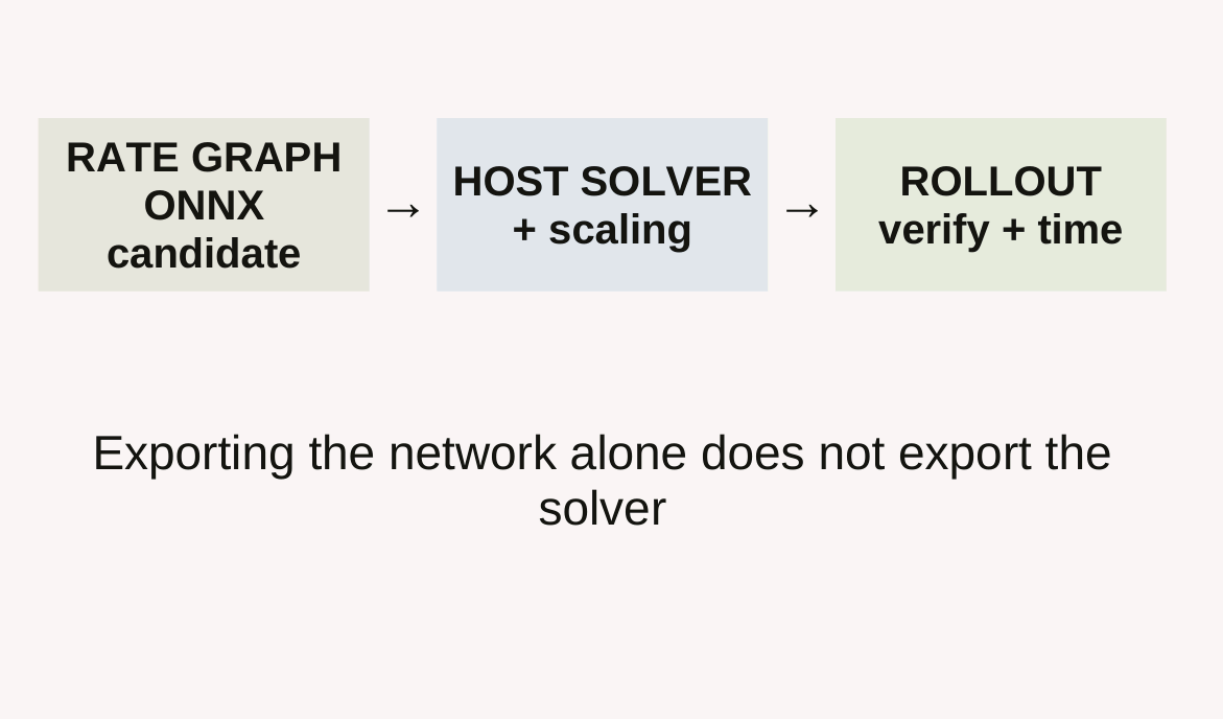
Sources: Q1, V1, E2. Full source details follow the glossary.

DEPLOYMENT

33. ONNX carries a computation graph

Export the right component and verify the complete inference path.

ONNX is an interchange representation for computation graphs. Exporting a rate network does not automatically export an adaptive ODE integrator, its stopping decisions or the complete training procedure. A practical design could evaluate an exported rate model inside a host solver, provided supported operators, precision and interfaces match. That is a proposed deployment route here, not an inspected existing vZero ONNX feature. Preserve normalizer parameters, units, species ordering and bounds. Compare full trajectories and meaningful physical outputs before benchmarking throughput and latency.



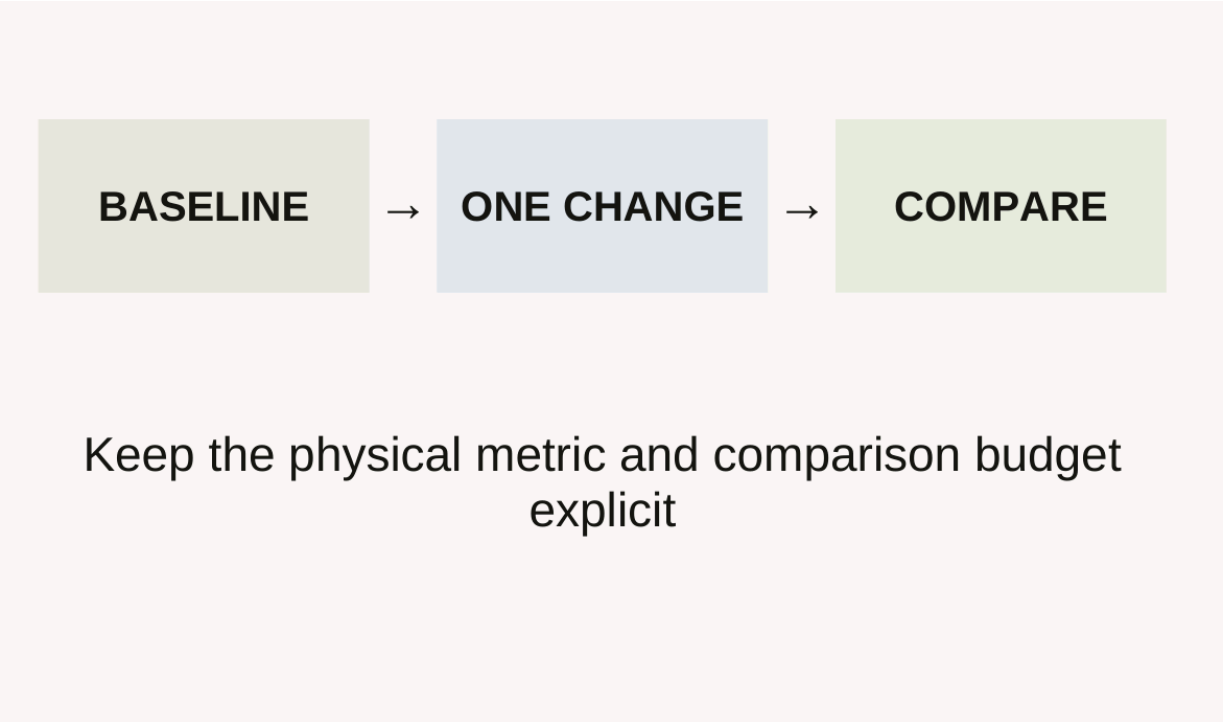
Sources: Q2, Q1, V1. Full source details follow the glossary.

THE NEXT EXPERIMENT

34. Change one layer, keep the comparison honest

A useful ablation answers a specific question.

An ablation is an experiment that isolates a design choice. Start with a verified reference and a small Neural ODE. Separately test channel scaling, solver/tolerance, objective aggregation, graph or latent representation, and deployment precision. Keep the split, physical reporting metrics and training budget fixed when they are not the variable being studied. Record function evaluations and wall time rather than asserting speed from network size alone. Use a pilot check to avoid spending the entire budget on a failed model, then evaluate held-out conditions. This is a proposed study plan, not a new chemical-model result.



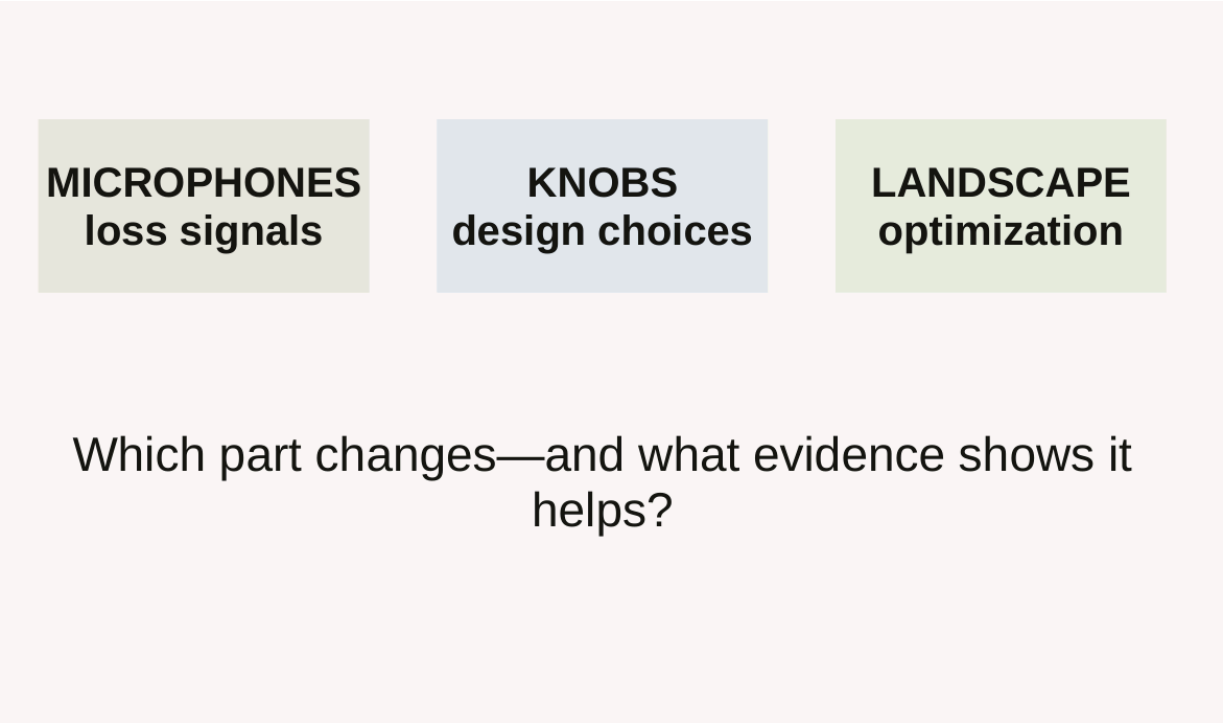
Sources: V1, V2, E1. Full source details follow the glossary.

RECAP

35. Learn the rate. Respect the solver. Check the physics.

The notes describe connected tools with distinct responsibilities.

Teach-back questions: What does the Neural ODE network output? Why can normalized loss improve while a trace-species trajectory remains poor? What extra condition made the three-variable Koopman toy exact? Why is a low-rank reconstruction not yet a predictive model? What must remain outside or alongside an exported ONNX rate network? Ask the learner to place each handwritten topic on the five-job map. Correct the connections before producing the requested LaTeX/TikZ version. The editable PowerPoint and companion guide provide the concrete content to review.



Sources: H1, E1, E2. Full source details follow the glossary.

Glossary

State — Variables describing the model at one instant.

Rate or derivative — How fast the state changes per unit time.

Vector field — A rule assigning a rate to each state.

Solver or integrator — Algorithm that accumulates rates to approximate a trajectory.

Parameter — Adjustable coefficient such as a network weight.

Loss or objective — Numerical score used to fit the model.

Gradient — Sensitivity of a scalar score to adjustable variables.

Adjoint — Auxiliary sensitivity system used to calculate gradients.

Jacobian — Matrix of local sensitivities of a vector-valued function.

Eigenvalue — A local linear rate along a corresponding mode; not a universal stiffness diagnostic by itself.

Observable — Function of the state, such as a sensor readout or x^2.

Closure — The chosen state/features contain what is needed to describe their evolution.

DMD — Dynamic mode decomposition: estimate a linear evolution map from snapshots.

POD — Proper orthogonal decomposition: represent snapshots with dominant modes.

SVD — Singular value decomposition: factor a matrix into paired directions and nonnegative strengths.

Latent state — Internal coordinates used by a model rather than directly observed variables.

Lagrangian — Mechanical energy function, typically kinetic minus potential energy.

Cholesky factor — Triangular factor used to construct or factor a positive-definite matrix.

Quantization — Represent values using a discrete set of levels.

ONNX — Open Neural Network Exchange, a model-graph interchange format.

Ablation — Controlled experiment isolating one design choice.

Corrections and sources

A finite-difference slope along a chemical trajectory is not automatically an eigenvalue of the full chemical Jacobian. It can motivate a stiffness investigation but cannot prove a particular RK4 stability violation by itself. The residual identity for affine scaling does not mean the network, optimizer and solver follow identical dynamics. A logarithm is not defined at zero, so claims about a log-transformed model there need a boundary policy. These qualifications are retained even where an older local document uses stronger wording.

N1 — Neural Ordinary Differential Equations, Chen et al. (2018)

https://arxiv.org/html/1806.07366v5

N2 — Stiff Neural Ordinary Differential Equations, Kim et al. (2021)

https://arxiv.org/abs/2103.15341

N3 — ChemNODE, Owoyele and Pal (2021)

https://arxiv.org/abs/2101.04749

N4 — torchdiffeq official implementation

https://github.com/rtqichen/torchdiffeq

K1 — Modern Koopman Theory, Brunton et al.

https://arxiv.org/html/2102.12086v1

K2 — Koopman Invariant Subspaces, Brunton et al. (2016)

https://arxiv.org/abs/1510.03007

D1 — On Dynamic Mode Decomposition, Tu et al.

https://arxiv.org/abs/1312.0041

L1 — Latent ODEs, Rubanova et al. (2019)

https://arxiv.org/abs/1907.03907

R1 — Deep Lagrangian Networks, Lutter et al.

https://arxiv.org/pdf/1907.04490

R2 — Lagrangian Neural Networks, Cranmer et al.

https://arxiv.org/abs/2003.04630