When can you differentiate coupled simulation components separately?

A cheap test, and a cold plate that answers it

Tesseract Hackathon 2026 · Track: multi-physics & coupled systems · Author: @TAUIL-Abd-Elilah · source · Apache-2.0

Abstract

Composing differentiable simulation components requires both component derivatives and an implicit system solve. We ask when the tempting shortcut— differentiating each component in isolation—is good enough.

Our natural-convection cold plate serves three active Tesseracts, spanning PyTorch, C++/Eigen and JAX or Fortran/Enzyme. Swapping the independently implemented thermal backends changes the end-to-end gradient by only 5.3×10^{-12}. Yet cutting the coupling loop causes 0.0003–86% error with no forward warning. Spectral radius $\rho(\Phi_T)$ is constant in one test where error moves 136-fold; the one-VJP, objective-aware residual $\gamma = \|\Phi_T^T g\|/\|g\|$ tracks error at 0.995 versus 0.825 for ρ. Across 2,377 synthetic fixed points the figures are **0.989 versus 0.691**, though γ predicts poorly for repelling loops.

The shortcut can still descend while ranking the fifty most influential cells at chance (Spearman -0.011). Using γ as a gate removes 92% of cross-boundary VJP calls while finishing within 0.51%, yet refuses the state with 115% error.

Most importantly, under one zero-sum raw-design rule the composed sensitivity delivers **58% more realised cooling** in a true re-solve. A retrospectively frozen extension retains **35 exact wins, 1 shortcut win, 3 ties and 9 noncomparable attempts** (35/39 comparable; 81.1% post-freeze descriptive seed-cluster-bootstrap lower endpoint).

1. The problem

Two solvers that feed each other do not form a pipeline. In our case a Stokes–Brinkman flow solver and an advection–diffusion thermal solver are coupled in both directions: buoyancy makes temperature drive the flow, advection makes the flow drive temperature. The steady state is a fixed point,

$T^* = \Phi(T^*, \theta)$, Φ = thermal(fluid(T)),

and the sensitivity of any objective $J(T^*)$ requires the implicit function theorem,

$dJ/d\theta = (\partial\Phi/\partial\theta)^T \lambda$, $(I - \Phi_T)^T \lambda = g$, $g = dJ/dT$. (1)

The shortcut treats the loop as feed-forward and uses $\lambda_0 = g$. Its residual in the exact adjoint equation is

$r_0 = g - (I - \Phi_T)^T\lambda_0 = \Phi_T^T g$, $\lambda - \lambda_0 = (I - \Phi_T^T)^{-1} r_0$. (2)

The first identity is exact whenever the derivatives exist; the second holds when the adjoint system is invertible. A Neumann expansion of the inverse would additionally require $\rho(\Phi_T) < 1$ and is not used at our repelling headline state.

Relation to existing work

TOFLUX (Padmanabha et al., arXiv:2508.17564, 2025) is an open-source JAX framework for differentiable thermo-fluidics. Alexandersen et al. (*Int. J. Heat Mass Transfer*, 2016, arXiv:1508.04596) solve 3D natural-convection heat-sink design at 40–330 million degrees of freedom; our 2D result reproduces their branching structure qualitatively. Padway and Mavriplis (*Numerical Algorithms*, 2021, arXiv:2104.02826) analyse fixed-point sensitivities under approximate linearisation; our loop-cut λ_0 is one such approximate adjoint and $\Phi_T^T g$ is its exact equation residual.

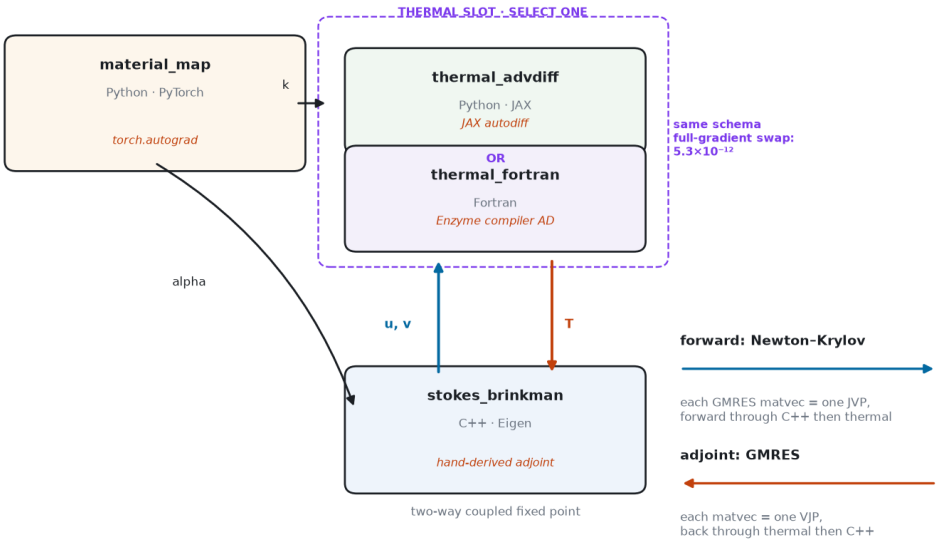
Our contribution is narrower: compose hand-adjointed C++, JAX, PyTorch and compiler-differentiated Fortran components; show $\rho(\Phi_T)$ constant while error moves 136-fold; and turn the objective-aware residual γ into a measured, single-VJP pipeline check.

2. The composition

Three Tesseracts are active in a run; the thermal slot selects one of two backends. Across the repository there are three implementation languages and four ways of producing a derivative:

component	language	derivatives from
`stokes_brinkman`	C++ / Eigen	hand-derived discrete adjoint (no AD tool)
`thermal_advdiff`	Python / JAX	`jax.jvp` / `jax.vjp`
`thermal_fortran`	Fortran	Enzyme, an LLVM compiler pass
`material_map`	Python / PyTorch	`torch.autograd`

On the headline `inertia=0` path, the fluid system is linear in w = (u,v,p), so `A(α) w = b(T)` and its exact JVP and VJP are extra solves against the same sparse LU—a transpose solve plus an analytic scatter, by hand. The Fortran component inverts that approach: flang emits LLVM IR, an Enzyme pass differentiates it, and $\partial R/\partial T$ is recovered *exactly* from nine Enzyme JVPs by a 3×3 colouring (the stencil is five-point, so cells whose indices agree mod 3 never interact).



Three implementation languages · four derivative stacks · three served components per run.

Matrix-free coupled adjoint: every Krylov matvec is a conversation between the active solvers. The thermal slot accepts either backend with no caller change; in the tested swap, the end-to-end gradients agree to 5.3×10^{-12}.

> **The composition, and where derivative information travels.** Each run serves the material map, fluid solver and one implementation in the thermal slot. The design flows into the two-way fluid–thermal loop; the adjoint travels back up it. Each box-boundary arrow is a container round-trip, crossed once per Krylov matvec.

Both halves of the gradient are Krylov solves whose matvecs cross the container boundary: Newton–Krylov forward, applying $(\Phi_T - I)$ via JVPs, and GMRES for the adjoint of (1), applying $(I - \Phi_T)^T$ via VJPs. At our operating point the loop gain exceeds one, so the fixed point is locally unstable under Picard iteration and our Picard runs fail; $\rho > 1$ alone does not rule out every specially chosen trajectory. Newton does not require Φ to be contractive; here the linearisation is nonsingular and damped Newton converges from the stated start. The JVP endpoints make that matrix-free forward solve robust.

A nonlinear component. `inertia = 1` adds steady convective acceleration and solves the now-nonlinear fluid block by damped Newton; zero retains the infinite-Prandtl Stokes path bitwise. Because $(u\cdot\nabla)u$ is bilinear and contains neither α nor T, the parameter scatters stay unchanged and the adjoint simply replaces A by the converged Jacobian. Against an autodiffed reference, forward error is $< 10^{-8}$, JVP/VJP error $< 10^{-7}$ and the adjoint identity $< 10^{-8}$; the full composition matches a coupled finite difference to 8.5×10^{-8}.

At $\mathrm{Ra} = 3 \times 10^4$, dropping inertia changes the measured design gradient by only **0.002% in water and 0.017% in air**, cosine 1 to eight decimals (`inertia_study.py`). This supports that tested comparison, not every regime.

Chain versus loop. The distinction matters and is often blurred. A *chain* — A feeds B feeds an objective — is differentiable by one sweep of the chain rule; a single `jax.grad` over wrapped components suffices and nothing is solved. This is a *loop*: the fluid solver's output is the thermal solver's input and vice versa, so no ordering makes one sweep sufficient. The steady state must be solved for, and its sensitivity requires a second, transposed solve whose operator this implementation applies matrix-free by calling both components.

Why not `jax.custom_vjp`? A custom backward rule could wrap this algorithm, including GMRES and repeated calls to external VJPs. It is an AD hook, not a packaging, schema, serving, lifecycle or isolation system. Here the kernels are C++/Eigen and flang/Enzyme-built Fortran; Tesseract keeps their toolchains and runtimes isolated, supplies uniform matrix-free JVP/VJP endpoints, and lets the thermal implementations swap under one schema. Recreating the RPC, lifecycle and dispatch inside a custom rule is possible, but bespoke.

Two claims are enforced by the build rather than asserted: neither the C++ nor the Fortran image can import `jax` , `torch` , `tensorflow` , `autograd` or `casadi` , and the Fortran library must contain `cosh` among its linked symbols — a function present in no source file, being Enzyme's generated derivative of the `tanh` in the Péclet weighting. Both are re-checkable from outside the build with `scripts/verify_integrity.sh` .

Components are interchangeable. `thermal_advdiff` and `thermal_fortran` implement the same equation behind the same schema. Swapping them (`compare_thermal_backends.py`) changes the component field by 7.1×10^{-16}, the JVP by 1.4×10^{-15}, the VJP by 4.3×10^{-15}, the converged coupled state by 4.8×10^{-12}, and **the end-to-end gradient by 5.3×10^{-12}**, cosine 1.000000000000. At these tested states, the backend swap leaves the gradient unchanged to numerical precision.

3. Validation

Classical critical Rayleigh number. Stokes flow is the infinite-Prandtl limit in which the classical onset result $\mathrm{Ra}_c = 1707.762$ is derived, so it is the correct benchmark for the original `inertia=0` path. A separate de Vahl Davis side-heated cavity check activates `inertia=1` at $\mathrm{Pr} = 0.71$ and therefore tests the nonlinear finite-Prandtl path on its own terms. At N=32, both cavity cases converge and all six Nusselt/centerline-velocity metrics are within **1.2%** of the published references. A separate exact-1 W SI audit is retained as a failed boundary: only **3 of 6** planned layout/mesh solves converged, the N=32 finned solve stalled, and even the converged baseline temperature is outside the constant-property liquid-water regime. We therefore make no dimensional resistance or fin-performance claim. At the conduction state the coupling loop *is* the linear stability operator, so onset occurs exactly where $\rho(\Phi_T) = 1$. Bisecting on Ra (`benchmark_critical_rayleigh.py`):

aspect ratio	1	2	4	8
Ra_c measured	2519.07	1970.84	1779.02	**1707.97**
excess	+47.5%	+15.4%	+4.2%	**+0.012%**

No-slip side walls stabilise a confined layer, so Ra_c must exceed the unbounded value and descend towards it as the box widens. It does, agreeing to four significant figures. One number checks the fluid solver, the thermal solver, the coupling between them and the loop-gain machinery simultaneously.

Order of accuracy. Richardson extrapolation on two independent grid trios gives observed order 1.87 and 1.83, with 0.02–0.05% error on the finest grid (`grid_convergence.py`).

Independent implementations agree. The composed pipeline reproduces a separately written monolithic reference to 1.5×10^{-12} on the converged coupled state, reached by a different nonlinear solver (5 Newton iterations against 21 Picard). Conduction-only energy balance closes to 5.3×10^{-15}.

The coupling-complete discrete gradient is validated. A directional derivative agrees with central differences to 7.45×10^{-6}, holding at $\sim10^{-5}$ across step sizes from 10^{-3} to 10^{-4} — that plateau is the differencing noise floor, since the fixed point is converged only to $\sim10^{-10}$.

4. What the shortcut costs

We compare against the *charitable* shortcut: it uses the fluid solver's adjoint in full and differentiates the whole chain ρ → α → (u,v) → T, and is exact except that it treats the temperature entering buoyancy as constant. It errs only by ignoring the loop.

At a strongly coupled state (Ra = 3 × 10^4) it carries **86% relative error**, cosine 0.53 against the true gradient, and the **wrong sign on 33% of design variables**. At the states our optimiser actually visits it is 4–20% off with cosine above 0.98. Same code and physics, but strong regime dependence: over the full coupling sweep the error grows from 3 × 10^{-6} to 0.86 while J moves by only 13%. **The forward solution gives no warning.**

Spatially, the error is not noise. At high coupling the coupling-complete gradient develops a coherent region of *positive* sensitivity — adding material there makes the chip hotter, because it obstructs the convection cell removing the heat — and the shortcut has no such region anywhere. The disagreement is one contiguous blob: an entire physical term, missing.

5. Predicting it

The obvious statistic fails. ρ(Φ_T) is the natural candidate and correlates respectably (0.825 with log error) when designs and Rayleigh numbers are varied together. But it is an objective-blind asymptotic modal rate: it does not encode how g aligns with the operator's modes. Two states at the same Ra have ρ = 0.682 with 40% error and ρ = 0.377 with 79% error: it orders them backwards.

A constant cannot explain a variable. The decisive experiment holds the design, the Rayleigh number and hence the entire coupled state fixed, and varies only the objective. ρ(Φ_T) is then *identical by construction*, while g = dJ/dT differs (`objective_sweep.py`):

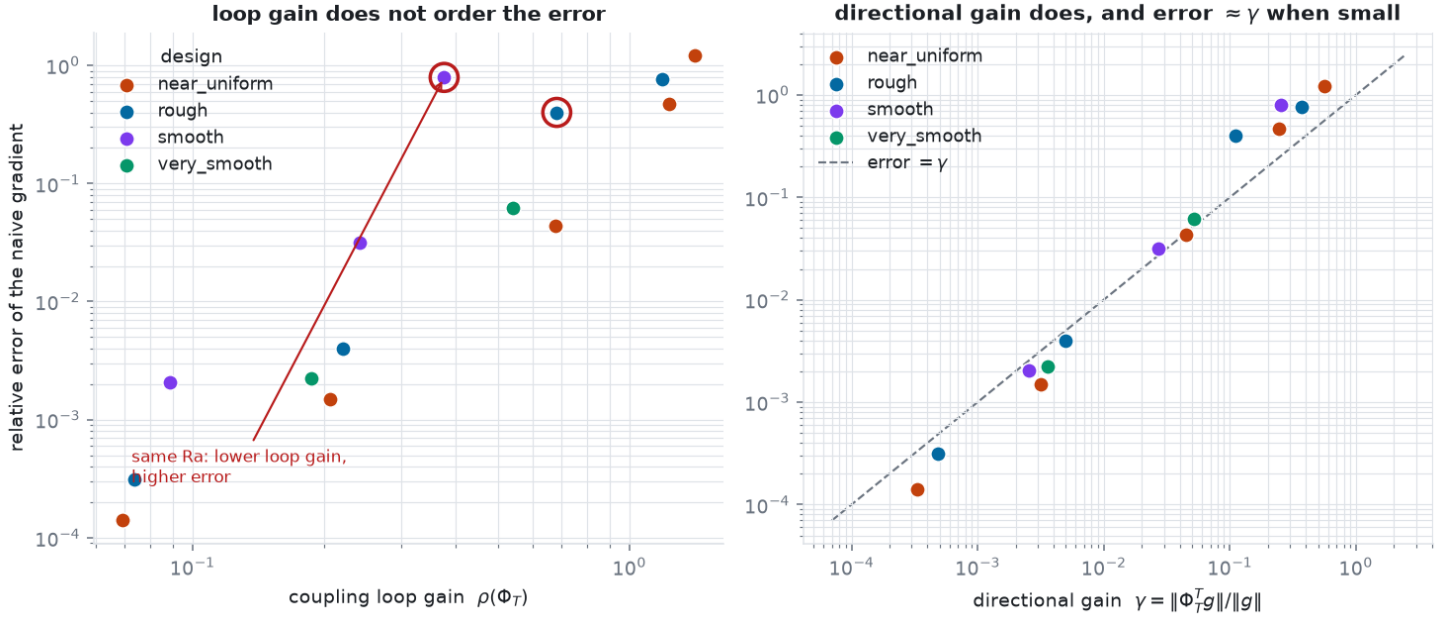
objective	outlet	top-half	chip peak	chip mean	domain	left column
γ	0.0027	0.0076	0.0261	0.0300	0.0967	0.3879
naive error	0.0117	0.0026	0.0385	0.0399	0.0211	0.3535

ρ(Φ_T) = 0.5481 on every column while the error varies **136-fold**. A constant cannot explain that spread: spectral radius alone is insufficient for objective-specific gradient error.

The directional gain. Equation (2) says the loop-cut adjoint has the exact residual $\Phi_T^{\mathsf{T}} g$, so the natural relative measure is

$$\gamma = \|\Phi_T^{\mathsf{T}} g\| / \|g\|, \quad (3)$$

one VJP through the loop. Across **14 converged configurations** drawn from four design families and five attempted Rayleigh levels, log γ correlates with log error at **0.995**, against 0.825 for ρ (`predict_error.py`). It remains 0.994–0.997 when each family is held out, with a seeded bootstrap 95% interval of 0.989–0.999 (`predictor_statistics.py`).

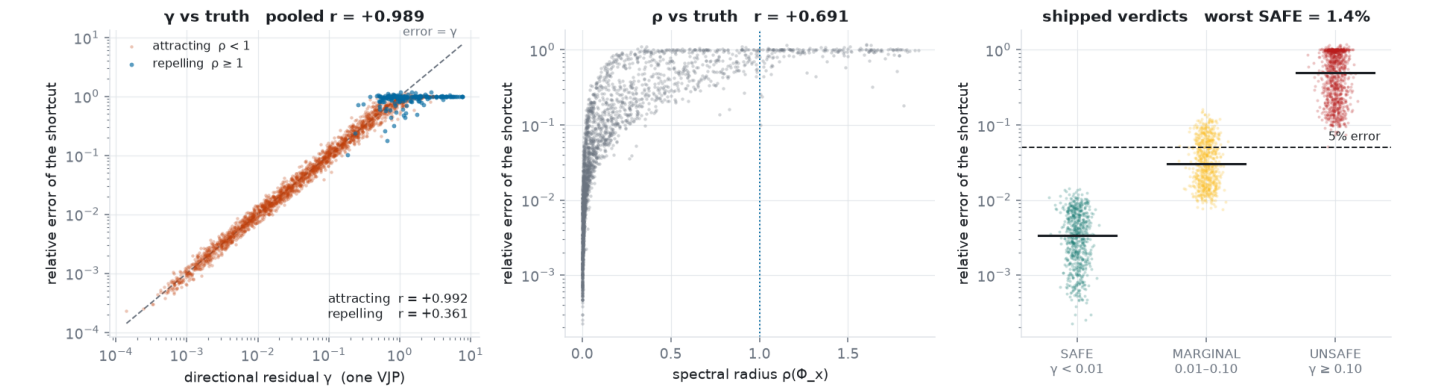


The directional gain predicts; the spectral radius does not. Each point is one (design, Rayleigh number) configuration, coloured by design family. Left: ρ(Φ_T) against the measured relative error of the component-wise gradient — log-log correlation 0.825, with visible order reversals, pairs where the configuration ρ calls safer is in fact the more damaged one. Right: γ against the same errors, correlation 0.995, tracking the dashed error = γ line while γ is small.

γ is a guide, not a formula. Across objectives it correlates at 0.80 and misses two rows above by about 4× in each direction — expected, since converting an adjoint residual to design-gradient error also depends on $(I-\Phi_T^{\mathsf{T}})^{-1}$ and on Φ_θ^{T}. On this benchmark γ < 0.01 accompanied roughly percent-level error and γ > 0.1 flagged danger. `coupling_check.py` exposes these as configurable, benchmark-calibrated defaults rather than universal guarantees.

Does it generalise? We removed the physics and sampled 2,377 closed-form fixed points (`gamma_generalization.py`): symmetric, non-normal, sparse and low-rank; linear and nonlinear; ρ from 10^{-3} to 1.9. Log γ correlates with error at **0.989** against **0.691** for ρ and wins in every family and both kinds. Nothing called SAFE exceeds 1.4% error; all 965 UNSAFE draws exceed 5%.

The boundary is equally important. Correlation is 0.993 for attracting loops but only **0.36 for repelling ones**, where the inverse in (2) may amplify the residual. We therefore compute the adjoint whenever the loop repels, including the headline state at ρ = 1.19; γ is not advertised as a theorem.

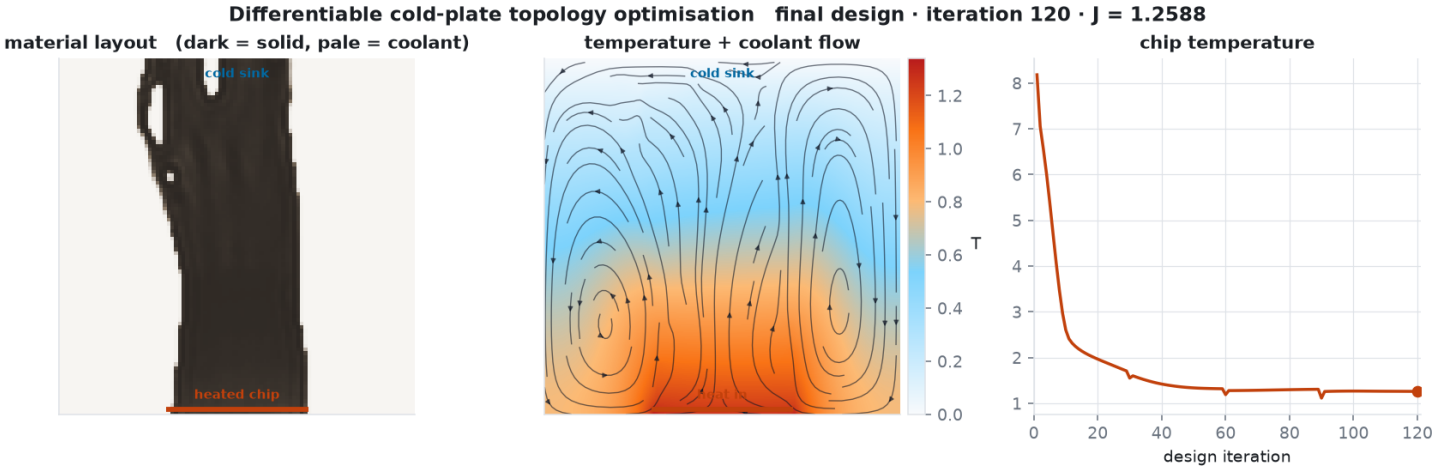


One VJP, no physics. Left: measured relative error of the component-wise gradient against γ on 2,377 randomly generated coupled fixed points, spanning four decades and hugging the identity error = γ; the repelling cases (blue) peel away and saturate, which is the documented limit. Centre: the spectral radius against the same errors — the obvious diagnostic, and visibly not a function of the thing it is meant to predict. Right: the thresholds this repository ships, with medians marked. Nothing γ called SAFE exceeded 1.4% error, and everything it called UNSAFE genuinely exceeded 5%.

Using it as a budget. At 48^2 over 80 iterations, a 0.10 gate sees $\gamma \in [0.020, 0.074]$, reduces cross-boundary VJPs from 1015 to 80 and finishes within 0.51% ($J = 1.3113$ versus 1.3180); no layout-equivalence claim is made. At $Ra = 3 \times 10^4$ it returns $\gamma = 0.404$ and refuses, against 115% measured error. One VJP thus separates these benchmark regimes even when forward histories do not.

6. The design problem

Minimising mean chip temperature subject to a 35% solid-material budget, with density filtering and Heaviside continuation, 120 iterations at 96×96: **J falls from 8.18 to 1.26, an 84.6% reduction**. The optimiser finds a branching tree that conducts heat toward the sink while leaving channels open for buoyancy to carry the remainder — the structure the natural-convection topology optimisation literature reports.



The converged cold plate. Material layout (dark = solid metal), the resulting temperature field with streamlines of the buoyancy-driven flow, and the objective history. The branching conductor reaches from the chip strip on the bottom wall toward the cold sink while leaving the convection cells room to circulate; both are needed, and the balance between them is what the coupled gradient is resolving.

7. What the gradient changes — and what it does not

We ran the long optimisation twice, once with each gradient, and **both succeeded**; the shortcut finished marginally lower (1.2576 against 1.2588). Adam normalises each coordinate, and along this weakly coupled trajectory the shortcut's cosine stays above 0.98. A successful optimiser is not evidence that its gradient is right.

We therefore tested a discrete engineering action at the strong state instead (`intervention_test.py`, 20^2, $Ra = 3 \times 10^4$). Each gradient receives the same zero-sum raw-design rule: increase its twenty most favourable raw variables and decrease its twenty least favourable by the same amplitude. We then discard both linear predictions and re-solve the true coupled forward problem:

raw-design step	ΔJ, composed choice	ΔJ, shortcut choice	more cooling
0.010	−0.01715	−0.01121	53%
0.025	−0.04322	−0.02800	54%
0.050	**−0.08789**	−0.05565	**58%**

The gradients agree on only 40% of the add set and 10% of the remove set. The composed choice wins all three fresh forward solves, delivering **58% more realised cooling** at the largest step for the same raw-variable count and amplitude. The nonlinear material map means this is not an equal realised physical-density budget.

A retrospectively frozen 16-seed × 3-Rayleigh extension retains all 48 attempts: **35 exact wins, 1 shortcut win, 3 ties and 9 noncomparable**, or 35/39 among comparable cases, with an **81.1%** post-freeze descriptive seed-cluster-bootstrap lower endpoint. Thirteen cells overlap prior evidence; among the other 35, the exact action leads in 24/28 comparable cases (one shortcut win, three ties). This is a robustness extension, not an independent confirmation set.

A frozen eight-step showdown also stopped when the composed step-six candidate failed to converge after five accepted decisions; the shortcuts completed eight. The incomplete primary endpoint is **not evaluable**, so there is **no eight-step endpoint verdict**. Post-hoc, the common five-step reductions are 11.83% composed, **5.16%** loop cut and **4.71%** frozen flow—not a protocol win.

In attribution (`sensitivity_ranking.py`), the shortcut keeps the true top-50 signs yet ranks magnitudes at chance (Spearman −0.011), misses 44%, and promotes cell #1016 of 1024. Limitations: this is one 2D application; the strong solve failed from the optimiser's near-uniform start; and repelling cases are outside γ's threshold calibration.