

LensIR: Compositional Reverse Semantics for a Typed Tensor Fragment

Anonymous

Version 1.2 - 12 October 2026 - Prepared for editorial review

Author: Anonymous

Abstract

We present a finite tensor language with explicit shapes, shared parameter identities, and residual reverse execution. Its ideal semantics is a category of smooth forward maps paired with linear cotangent responses. We prove type preservation, compositional residual correctness, reverse correctness for straight-line programs, and conditional backend simulation. The central sharing rule is the dual of the diagonal: cotangents from repeated occurrences must be added. A tied-weight residual block supplies a worked example and a NumPy implementation, with an independent closed-form pullback and directional finite-difference checks. Exact derivatives over real spaces, declared training responses, and floating-point evaluations receive distinct contracts. This note supplies a compact compiler target and executable reference for the declared fragment; it does not assert coverage of arbitrary architectures, source languages, effects, or accelerator backends.

1. Scope and categorical precedents

Compositional learning [1], reverse derivative categories [2], and lens-based gradient learning [3] establish the categorical background. We use their concrete smooth-map setting to state a compiler contract that includes shapes, residuals, and sharing. Reverse correctness is the chain rule expressed for this language, rather than a new differentiation principle. Generalized lenses [4] are a related construction; the response pairs below are not automatically delta lenses or lawful database update lenses.

All ideal scalar values are real numbers. A shape $s = (n_1, \dots, n_k)$ with positive integer dimensions denotes the finite-dimensional vector space

$$T_s = \mathbb{R}^{n_1 \times \dots \times n_k}.$$

The empty shape denotes $\mathbb{R}$. Arrays carry the Euclidean pairing obtained by summing entrywise products, identifying T_s^* with arrays of the same shape. This identification is declared coordinate structure, not an intrinsic identification on every manifold.

Notation and proof contract. A bar denotes an accumulated cotangent, Df_x^* the dual derivative, $\odot$ entrywise multiplication, and T matrix transpose. A response is semantic data; a residual is implementation data. Sections 2-4 prove exact real-arithmetic statements; Sections 5-6 separately state backend hypotheses and finite numerical evidence.

2. Response semantics

Definition 2.1 (Response map). A response map $L : X \rightarrow Y$ between finite-dimensional real vector spaces is a pair (f, b) , where $f : X \rightarrow Y$ is smooth and $b : X \times Y^* \rightarrow X^*$ is smooth and linear

in its second argument. Its forward and backward observations are both part of the semantics. For $M = (g, c) : Y \rightarrow Z$, set

$$M \circ L = (g \circ f, d), \quad d(x, \gamma) = b(x, c(f(x), \gamma)).$$

The identity is $(\text{id}, (x, \alpha) \mapsto \alpha)$.

Proposition 2.2 (Category and exact derivative interpretation). Response maps form a category. Sending f to $(f, (x, \beta) \mapsto Df_x^* \beta)$ preserves identity and composition.

Proof. Both bracketings of a triple composite give

$$b(x, c(f(x), e(g(f(x)), \delta))).$$

Identities substitute the seed unchanged. Linearity of the backward composite follows from linearity of each factor. The derivative of an identity is the identity; the chain rule gives $D(gf)_x^* = Df_x^* Dg_{f(x)}^*$. $\square$

Independent products use componentwise responses. Copying a value $\Delta : X \rightarrow X \times X$ has response $(\alpha, \beta) \mapsto \alpha + \beta$. Discarding a value has zero response. These rules account for graph fan-out and unused inputs.

Definition 2.3 (Residual implementation). An implementation of (f, b) consists of a residual set R and maps

$$\text{fwd} : X \rightarrow Y \times R, \quad \text{rev} : R \times Y^* \rightarrow X^*,$$

such that $\text{fwd}(x) = (f(x), r_x)$ and $\text{rev}(r_x, \beta) = b(x, \beta)$. Residuals store information sufficient for the selected response.

Lemma 2.4 (Residual composition). Residuals $R_L \times R_M$ implement $M \circ L$ by forward execution in order L, M and reverse execution in order M, L .

Proof. Forward execution produces $f(x)$, then $g(f(x))$, retaining r_L, r_M . Reverse execution returns $b(x, c(f(x), \gamma))$, exactly Definition 2.1. Products pair residuals and responses. $\square$

If a smaller residual can reconstruct a sufficient original residual without changing observations, the reconstruction followed by the original reverse map has the same response. This is the precise contract for checkpoint replacement in the pure fragment. Replaying effects or random streams needs additional contracts.

3. A finite typed language

Definition 3.1 (Module). A module has finitely many input names with shapes, a topologically ordered list of nodes, and a nonempty ordered list of outputs. A node has a fresh name, a primitive, a list of previously available operand names, and any primitive-specific attributes. Inputs designated as parameters are ordinary named inputs: every use of one name reads one value. Repeated operand occurrences are permitted and remain separate uses.

The fragment has these primitives:

- **add, mul**: two operands of the same shape s ; entrywise addition or multiplication.
- **matmul**: shapes (a, b) and (b, c) yield (a, c) by matrix multiplication.
- **tanh**: entrywise hyperbolic tangent, preserving shape.
- **sum**: any tensor shape yields a scalar by summing all entries.
- **broadcast_rows**: shape (d) yields (B, d) by explicitly copying the vector to B rows.

Broadcasting is explicit. The node registry rejects unknown primitives, mismatched shapes, invalid dimensions, duplicate definitions, and use before definition. There is no implicit state, RNG, host call, branch, or loop in this fragment.

Definition 3.2 (Typing). A shape environment Γ assigns types to available names. A node is well typed when its operands are in Γ and the primitive’s shape rule applies. Extend Γ with the inferred result shape, requiring a fresh name. A module is well typed when all nodes and outputs pass these checks. Its ideal denotation $\llbracket P \rrbracket$ is a smooth map between the products of its input and output spaces.

Theorem 3.3 (Type preservation and termination). A well-typed module evaluated with matching input shapes produces values of every inferred node and output shape and terminates in the ideal interpretation. A runtime enforcing the declared float64 shape contract preserves those shapes.

Proof. Induct through the finite node list. Input shapes are the premise. Each primitive’s formula has its inferred shape, so extending the value environment preserves the induction invariant. Output lookup uses existing names. Finitely many total ideal primitive evaluations terminate. The runtime statement uses the same induction and its primitive shape checks; it makes no assertion about unbounded physical resources. $\square$

4. Reverse transformation and shared parameters

Forward execution saves the values needed by each primitive response. Reverse execution initializes every cotangent to zero, adds each supplied output seed to its named output, then traverses nodes in reverse order. Each local response is added to every operand occurrence. Thus duplicate output names and repeated operands accumulate rather than overwrite contributions. In pseudocode the invariant is:

```
cot[name] = zero(shape[name]) for every available name
for each (output occurrence, seed): cot[output] += seed
for node in reverse topological order:
    for each (operand occurrence, contribution) in local_response(node, cot[node]):
        cot[operand] += contribution
return cotangents of the named inputs
```

The input tape must come from the same module and forward evaluation, with saved primal values retained unchanged until the reverse pass. The reference implementation supplies a full value tape; residual minimality and validation of arbitrary externally supplied tapes are separate problems.

For elementwise multiplication, matrix multiplication, and row broadcasting, the local responses are respectively

$$(x, y, \bar{z}) \mapsto (\bar{z} \odot y, \bar{z} \odot x),$$

$$(A, B, \bar{C}) \mapsto (\bar{C}B^T, A^T\bar{C}), \quad (b, \bar{Y}) \mapsto \sum_{r=1}^B \bar{Y}_{r,:}.$$

Addition returns its seed to both occurrences; `tanh` multiplies by $1 - \tanh^2(x)$; `sum` broadcasts the scalar seed to all entries. These are their exact ideal pullbacks.

Theorem 4.1 (Reverse correctness). For every well-typed module in this fragment, matching input x , and matching output cotangent β , ideal residual reverse execution returns

$$\text{Reverse}_P(r_x, \beta) = D\llbracket P \rrbracket_x^* \beta.$$

Proof. Give each input an arbitrary tangent perturbation. Propagate it through the forward graph by the ordinary derivative rules. Initially the pairing of the output seeds with the output perturbations is the pairing $\langle \beta, D[P]_x \delta x \rangle$.

When eliminating a reverse node $v = o(x_1, \dots, x_k)$, substitute $\delta v = \sum_i D_i o \delta x_i$ into its current seed pairing. The contribution becomes $\sum_i \langle D_i o^* \bar{v}, \delta x_i \rangle$. Adding those terms to the accumulated operand cotangents preserves the total pairing of the remaining variables with their perturbations. If an operand is repeated, its contributions add in that same variable. Reverse topological order guarantees that all uses of a node have contributed before it is eliminated. After the finite traversal only input pairings remain. Equality of the pairings for every input perturbation identifies the returned cotangent with the displayed dual derivative. $\square$

Corollary 4.2 (Sharing rule). If $f(\theta, x) = \tilde{f}(\theta, \dots, \theta, x)$ with m occurrences, its parameter cotangent is $\bar{\theta} = \sum_{j=1}^m \bar{\theta}_j$.

Proof. The reuse map is the diagonal Δ_m . Its dual adds the m cotangents. Apply Proposition 2.2. $\square$

Proposition 4.3 (Declared responses). If a primitive is given a response satisfying its residual implementation equations but not equal to its derivative, the same transformation preserves the compositional declared response. It need not compute the classical derivative.

Proof. Interpret the graph in the response category, using copying and addition of cotangents at fan-out. Lemma 2.4 establishes the local residual equations under composition. The reverse elimination calculation composes those specified linear responses, whether or not they equal dual derivatives. $\square$

A forward identity with a zero response is a concrete example. Forward equality therefore does not determine the response contract. Nonsmooth primitives and custom training estimators require separate rules and adequacy claims; none is included silently in Theorem 4.1.

5. Backend contracts and numerical observations

Definition 5.1 (Backend simulation contract). For each shape, choose relations between ideal values and backend values, and between ideal and backend cotangents. Require each lowered primitive to preserve those relations in forward execution and its declared response. Require zero cotangents and cotangent addition to preserve them. Related inputs and seeds must have the corresponding types. Saved residuals must retain the data used by these local simulations. The relations must hold for every reachable primal, residual and seed, including all repeated-use accumulations. A quantitative simulation may need an explicit error budget in its relation; a fixed tolerance generally is not closed under unrestricted addition.

Theorem 5.2 (Conditional backend simulation). If a backend satisfies Definition 5.1 for every reachable primitive, lowering a well-typed module preserves the corresponding forward and reverse relations.

Proof. Forward induction relates every node value and its saved residual. In the reverse traversal, related output seeds and zeros initialize the induction invariant. Each local response preserves related cotangents by the primitive contract, and accumulation preserves the relation by the addition contract. Induction in reverse topological order reaches all inputs. $\square$

This is conditional correctness, not a verified backend contract for every library. The supplied implementation runs NumPy float64; the written ideal proof uses real arithmetic. Passing numerical checks does not prove a uniform rounding-error bound.

Lemma 5.3 (Sequential error propagation). For sequential stages with ideal maps f_i that are L_i -Lipschitz and approximations with uniform local error ε_i on all intermediate values encountered, let $e_0 = 0$. Then

$$e_k \leq \varepsilon_k + L_k e_{k-1}, \quad e_n \leq \sum_{i=1}^n \varepsilon_i \prod_{j=i+1}^n L_j.$$

Proof. Insert the ideal stage evaluated at the approximate input, apply the triangle inequality, then its Lipschitz bound. Induction expands the recurrence. $\square$

A reverse error contract must also include the dependence of local pullbacks on perturbed primal values. A tolerance relation is not automatically transitive and does not become an exact categorical equivalence.

6. Worked tied-weight residual block

Let $X \in \mathbb{R}^{B \times d}$, $W \in \mathbb{R}^{d \times d}$, and $b \in \mathbb{R}^d$. Broadcast b explicitly and define

$$A = XW + b, \quad H = \tanh A, \quad Y = X + HW + b.$$

There is one W and one b , each reused twice. For seed G at Y , direct differentiation gives

$$\bar{H} = GW^\top, \quad \bar{A} = \bar{H} \odot (1 - H \odot H),$$

$$\bar{X} = G + \bar{A}W^\top, \quad \bar{W} = H^\top G + X^\top \bar{A},$$

$$\bar{b} = \sum_{r=1}^B G_{r,:} + \sum_{r=1}^B \bar{A}_{r,:}.$$

Derivation. The final residual sum contributes G directly to X . The second matrix product contributes $GW^\top$ to H and $H^\top G$ to the second use of W . Differentiating $\tanh$ gives $\bar{A}$. The first product contributes $\bar{A}W^\top$ to X and $X^\top \bar{A}$ to the first use of W . Each bias occurrence contributes a row sum; Corollary 4.2 adds the shared contributions.

The serialized companion specializes $B = 2$, $d = 3$ and explicitly contains row broadcasting. Across 32 tied-block cases with batch sizes 1-3 and widths 1-4, generated pullbacks agree with the independent formulas to maximum absolute error 4.45×10^{-16} . All 96 directional finite-difference checks pass using step 10^{-6} and tolerance 3×10^{-7} ; maximum observed absolute discrepancy is 4.45×10^{-10} . Additional cases exercise repeated operands, repeated outputs, unused inputs, products, reductions, 10 invalid module rejections, and two runtime shape rejections. These rounded-up observed errors are for the source run (Python 3.12.14 and NumPy 2.3.5) and the revision run (Python 3.12.3 and NumPy 1.26.4), seed 20261012. Program version 1.0.0, exact measured errors, and hashes are recorded in `verification-results.json`.

Reproduction and data. The companion code/`tied_residual.json` supplies the $B = 2$, $d = 3$ module. Run `python3 code/lensir_example.py` for its numerical and shape checks, or `python3 code/run_checks.py --output verification-results.json` for the complete bundle. The deterministic random seed is 20261012. `parameters` identifies ordinary named inputs; it does not allocate a second value at each use or perform an optimizer update. NumPy evaluation is observational evidence for the supplied examples, not an implementation of exact arithmetic.

7. Shape families and limits of coverage

For a set V of satisfying shape valuations, regarded as a discrete category, $\text{Fun}(V^{\text{op}}, \text{Set}) \simeq \prod_{\nu \in V} \text{Set}$ is a presheaf topos. A tensor type is a family of value sets over V . A uniformly well-typed program

gives a morphism between such families; its ideal derivative interpretation is additional fiberwise smooth structure. Replacing sets by spaces gives a presheaf infinity-topos [5], but no extra numerical primitive or differentiation rule follows from doing so.

Architecture coverage is relative: every finite straight-line presentation over this primitive registry elaborates to the language by naming each intermediate value in dependency order. This does not imply that every named neural architecture uses only these primitives. Arbitrary effects, branching, loops, external kernels, source importers, and accelerator execution are beyond this note’s verified fragment. Their inclusion requires syntax, typing, residual and observation contracts, and appropriate extensions of the proofs.

Automation disclosure. OpenAI ChatGPT/Codex assisted with drafting, proof review, references, and implementation. The submission byline is Anonymous; no affiliation or independent referee approval is asserted. The general proofs are written arguments; the floating-point checks are bounded evidence, not formal proofs of all inputs or backend realizations.

References

1. Brendan Fong, David I. Spivak, Rémy Tuyéras. **Backprop as Functor: A compositional perspective on supervised learning**. LICS 2019. [arXiv:1711.10455](#).
2. Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin, Dorette Pronk. **Reverse derivative categories**. CSL 2020, extended version. [arXiv:1910.07065](#).
3. G. S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, Fabio Zanasi. **Categorical Foundations of Gradient-Based Learning**. 2021. [arXiv:2103.01931](#).
4. David I. Spivak. **Generalized Lens Categories via functors $\mathcal{C}^{\text{op}} \rightarrow \text{Cat}$** . [arXiv:1908.02202](#).
5. Jacob Lurie. **Higher Topos Theory**. Princeton University Press, 2009. [Electronic edition](#).