

A Forward-Pullback Tensor Language: Architecture-Relative Completeness and Homotopical Contracts

Anonymous author

21 October 2026 · Version 1.0

Abstract

A universal operator claim for neural networks combines several independent obligations: representation of architectures, closure under differentiation, correctness of the generated pullback, and correctness of numerical execution. We separate these obligations in a shape-indexed finite tensor language with explicit residuals. We prove compositional forward/pullback correctness and a constructive representation theorem for a grammar containing dense layers, finite convolutions, residual and parallel constructions, finite recurrence with shared parameters, normalization, and attention. Nonsmooth rules, dynamic traces, state and randomness receive separate contracts. Exact compiler refinement and a conditional numerical error recurrence identify what a backend must additionally establish. Residual re-encodings form groupoids whose nerves retain nontrivial implementation symmetries. In a presheaf infinity-topos, effective descent characterizes which implementation policies can be recovered from forward observations. A small NumPy reference interpreter supplies explicit JVPs, VJPs, architecture lowerings, serialization, and a structural rewrite, with deterministic exact and numerical checks. The correctness proofs are mathematical; the implementation is not certified by a proof assistant, and no coverage or numerical certification of all existing neural-network software is claimed.

1. Scope and relation to existing work

A backward program is part of the semantics of a trainable system. Observing its reported loss can reveal less than observing its exact forward function, and observing that function can reveal less than observing its resource use or state effects. A useful operator specification must say precisely which observation it preserves.

Reverse derivative categories axiomatize reverse differentiation and relate it to forward differentiation with a dagger on linear maps [1]. CHAD supplies compositional, type-respecting automatic differentiation and logical-relations correctness proofs [2]. Its extension treats total languages with richer type formers [3]. Efficient CHAD analyzes implementation complexity and supplies an Agda formalization of its complexity proof [4]. Our elementary chain-rule argument is an instance of this established pattern, not a new general theorem about automatic differentiation.

The contribution is a unified specification: a concrete tensor inventory, constructive lowerings for an explicit architecture grammar, a separation of derivative and execution contracts, and a homotopical model of residual implementations and observable policies. The artifact implements the finite tensor fragment. The compiler-refinement, numerical-bound and infinity-topos constructions are mathematical specifications and are not implemented proof checkers.

We use **architecture-relative completeness** to mean that every expression generated by a stated grammar has a translation into the tensor language with the stated semantics. This is distinct from approximation universality, equational completeness, Turing completeness, or coverage of every operator

accepted by an existing framework. A finite list of operator *schemas* also permits arbitrary finite shapes, index lists and constants; it is not a finite list of all operator instances.

2. Tensor types, domains and cotangents

Definition 2.1 (types). A shape is a finite list $s = (d_1, \dots, d_k)$ of positive integers; the empty shape denotes a scalar. Set

$$T_s = \mathbb{R}^{d_1 \cdots d_k}, \quad A, B ::= 1 \mid T_s \mid A \times B.$$

The terminal type 1 is the zero-dimensional vector space. Each type denotes a finite-dimensional Euclidean vector space with its usual pairing. Its linear dual is denoted A^* . Coordinates identify A^* with an equal-shaped array, but the dual role remains part of the specification. Product duals are identified with products of duals.

Definition 2.2 (smooth program). A smooth program $f : U \rightarrow B$ has an open domain $U \subseteq A$ and is C^∞ . Restrict domains as needed for composition. The first-order results below also hold for C^1 maps. Statements involving a full reverse differential category use the smooth class, so that iterated differentiation is available.

The forward directional and reverse operators are

$$\text{jvp}_f(a, v) = Df(a)v, \quad \text{pb}_f(a, u) = Df(a)^*u.$$

They satisfy

$$\langle \text{pb}_f(a, u), v \rangle = \langle u, \text{jvp}_f(a, v) \rangle. \quad (1)$$

The adjoint is attached to the linear derivative at a . An arbitrary nonlinear map does not itself acquire a linear adjoint. Trainable parameters belong to the input context: $f(\theta, x)$ is differentiated jointly unless the specification explicitly fixes a coordinate.

3. A finite tensor language

Definition 3.1 (SSA syntax). A context is a finite ordered list of tensor variables. A program has finitely many instructions

$$z_i := g_i(z_{a_{i1}}, \dots, z_{a_{ir_i}}), \quad a_{ij} < i,$$

followed by an output projection or tuple. Inputs precede instructions. Every operator has a shape rule, a real denotation, a domain condition and a pullback contract. Reusing a variable reuses the same context coordinate. Broadcasting is an explicit structural map. There is no implicit cast, shape change or custom-call escape.

The primitive schemas are constants; equal-shape addition and elementwise multiplication; reshape and axis permutation; fixed-index gather; sum reduction; concatenation; rank-two matrix multiplication; and elementwise exp, tanh, reciprocal, and inverse square root. Reciprocal requires nonzero inputs and inverse square root requires positive inputs. The inverse image of these open conditions under earlier instructions defines the program's open admissible domain. It can be empty.

Higher-rank contractions can be expressed by fixed reshapes, permutations, gathers, products and sums. Rank-two multiplication is retained as a useful computational primitive rather than expanded into scalar operations. Constants and finite index lists are parameters of schemas.

Lemma 3.2 (structural adjoints). For a fixed linear structural map L , its pullback is $L^\top$. In particular,

$$\Delta(x) = (x, x), \quad \text{pb}_\Delta(x, (u, v)) = u + v; \quad \text{pb}_\dagger(x, 0) = 0.$$

For $G_i(x)_j = x_{i_j}$, including repeated indices,

$$(G_i^\top u)_k = \sum_{j:i_j=k} u_j. \quad (2)$$

Proof. Expand $\langle u, Lv \rangle$ coordinatewise. The coefficient of each v_k is the corresponding coordinate of $L^\top u$. For gather, repeated occurrences contribute distinct summands. Copy is the repeated-index special case, and discard has zero codomain. $\square$

The inverse of a reshape or permutation is its adjoint under the chosen coordinate pairing. Reduction and broadcast are adjoint, while concatenation and splitting are adjoint. Indices in (2) are fixed discrete data; (2) differentiates values, not indices.

Lemma 3.3 (numerical primitive rules). The following pullbacks are correct over their real domains. The seed is u and $\odot$ denotes coordinatewise multiplication.

Forward map	Saved residual	Pullback
$x + y$	none	(u, u)
$x \odot y$	(x, y)	$(u \odot y, u \odot x)$
AB	(A, B)	$(uB^\top, A^\top u)$
$\exp x = y$	y	$u \odot y$
$\tanh x = y$	y	$u \odot (1 - y^2)$
$x^{-1} = y$	y	$-u \odot y^2$
$x^{-1/2} = y$	y	$-\frac{1}{2}u \odot y^3$

Proof. Scalar rules follow by differentiation and coordinatewise pairing. For multiplication, $D(AB)[H, K] = HB + AK$ and

$$\text{tr}(u^\top (HB + AK)) = \langle uB^\top, H \rangle + \langle A^\top u, K \rangle.$$

Addition and elementwise multiplication are the corresponding scalar computations. $\square$

4. Residual contracts and compositional AD

Definition 4.1 (certified mathematical tape). A tape contract for $f : U \subseteq A \rightarrow B$ consists of a finite-dimensional residual space R , smooth maps

$$F : U \longrightarrow B \times R, \quad K : R \times B^* \longrightarrow A^*,$$

and a proof of

$$F(a) = (f(a), r_a) \implies K(r_a, u) = Df(a)^*u \quad \text{for all } u. \quad (3)$$

Residual spaces are nonempty open subsets of Euclidean spaces, including the zero-dimensional terminal space. Require $K(r, -)$ to be linear, including at unreachable residuals. Correctness (3) concerns residuals actually generated by F . The word certified here describes a mathematical contract with a proof; it does not assert a machine-checked certificate in the artifact.

For contracts (F_f, K_f, R_f) and (F_g, K_g, R_g) , set

$$\begin{aligned} F_{gf}(a) &= (c, (r, s)), & (b, r) &= F_f(a), & (c, s) &= F_g(b), \\ K_{gf}((r, s), w) &= K_f(r, K_g(s, w)). \end{aligned} \quad (4)$$

Theorem 4.2 (composition and products). Formula (4) is a correct tape contract for $g \circ f$. Products of contracts are correct componentwise. Associativity and unit laws hold under the canonical re-association and deletion of unit residual factors.

Proof. The forward equation follows by substitution. On reachable residuals,

$$K_f(r, K_g(s, w)) = Df(a)^*Dg(f(a))^*w = D(g \circ f)(a)^*w.$$

For products, derivatives are block diagonal and pairing separates into the sum of the component pairings. Residual re-association changes neither evaluation order nor the nested expression for K . Identity uses a terminal residual and $K(*, u) = u$. $\square$

Theorem 4.3 (finite graph correctness). A well-typed finite SSA program $p : U \rightarrow B$ assembled from correct smooth primitive contracts has a well-typed tape transformation with

$$\text{Fwd}_p(a) = (\llbracket p \rrbracket(a), r_a), \quad \text{Rev}_p(r_a, u) = D\llbracket p \rrbracket(a)^*u. \quad (5)$$

Its forward directional transformation computes $D\llbracket p \rrbracket(a)v$.

Proof. Interpret instruction i as an extension of the current context,

$$H_i(z) = (z, g_i(\pi_i z)).$$

This is composition of projection/copy, the primitive, and pairing. Its reverse map keeps the cotangent for the retained context and adds the primitive's contributions into the selected coordinates. If two argument positions select one coordinate, both contributions are added. The full denotation is an output projection after $H_n \cdots H_1$. Apply Theorem 4.2 and Lemma 3.2 inductively, then transpose the output projection. The same induction with the ordinary chain rule proves the JVP statement. $\square$

Thus parameter tying is handled by accumulation into one parameter coordinate, rather than by differentiating independent copies and forgetting to identify them. Equation (1) follows for the transformed whole program. Agreement between a JVP and a VJP alone is insufficient evidence of derivative correctness: two incorrect transformations can still be transposes of each other.

Corollary 4.4 (conditional work and storage bound). Suppose each primitive forward and pullback runs in at most a fixed constant times its specified work, including reading/writing its arrays. If their total forward work is W and the total input size is I , the store-all transformation has work $O(W + I)$ and storage $O(S)$, where S is the total number of stored primal and cotangent coordinates. Initialize each cotangent slot once and accumulate directly into it.

Proof. Execute each primitive once in each pass. There are at most a fixed number of updates per argument occurrence, counted in primitive work. Initialization is linear in the stored coordinates, which are bounded by input size plus the counted writes. $\square$

This is a finite graph cost model. It does not prove cache efficiency, parallel span, distributed communication complexity or an optimal checkpoint schedule. Saved residuals can be reduced or recomputed only with an additional contract preserving the paired semantics.

5. Constructive architecture-relative completeness

Definition 5.1 (architecture grammar). Let $\mathcal{A}$ be the least grammar closed under typed composition, parallel product/concatenation, explicit sharing and the following constructors:

1. Affine dense maps and pointwise activations generated by the smooth primitive library.
2. Finite-dimensional, finite-stencil convolutions with fixed stride, padding, dilation and grouping data, and fixed sum/average pooling.
3. Residual addition of equal-shaped branches.
4. A fixed finite number T of recurrent steps with specified sharing of parameters and an explicit initial state.
5. Mean/variance normalization with a fixed $\epsilon > 0$, optional affine parameters, and explicit reduction axes.
6. Scaled dot-product attention, any fixed finite number of heads, fixed finite sequence lengths, concatenation and affine projections.

All types, masks and structural indices are fixed for a given grammar instance. A structural attention mask specifies nonempty allowed key sets and normalizes over those finite sets; it does not introduce a $-\infty$ real constant. The theorem applies on the open admissible domain of the lowered program. Fixed scalar scales and divisors are constants. Finite compositions of sigmoid can be expressed using exp, addition and reciprocal; activations outside the library require an extension contract.

Lemma 5.2 (constructive lowerings). Every constructor of $\mathcal{A}$ has a finite lowering into Section 3.

Proof. Dense maps are contraction plus bias broadcast. A finite convolution gathers input patches, inserts fixed padding constants, reshapes its weights and contracts; groups select separate channel blocks, and fixed stride/dilation change index lists. Pooling uses gather and reduction. Residual and parallel cases use addition and concatenation. A recurrence is unfolded T times while every parameter occurrence refers to the same input slot.

For each normalization slice of size m , use

$$\mu = m^{-1} \sum_j x_j, \quad c_j = x_j - \mu, \quad v = m^{-1} \sum_j c_j^2, \quad y_j = c_j(v + \epsilon)^{-1/2}. \quad (6)$$

Since $v \geq 0$ over the reals, the inverse-square-root condition is satisfied. Affine parameters add multiplication and broadcast. Attention uses

$$S = d^{-1/2} QK^\top, \quad P_{ij} = \frac{e^{S_{ij}}}{\sum_\ell e^{S_{i\ell}}}, \quad Y = PV. \quad (7)$$

Every denominator is positive. For a structural mask, gather each allowed row set, compute its normalization and multiply by its selected value rows. Sum the finite contributions. Heads are finite parallel copies with their stipulated parameter sharing. $\square$

Theorem 5.3 (architecture-relative completeness). For every typed $N \in \mathcal{A}$, one can construct a finite typed tensor program $L(N)$ with equal ideal real denotation. Its tape transformation satisfies (5), including gradients for shared parameters and differentiable initial state.

Proof. Structural induction on the finite construction of N , using Lemma 5.2 at the generating cases. Composition and parallel rules preserve types and denotation. Repeated uses of a context coordinate preserve sharing by construction. Lemmas 3.2 and 3.3 plus Theorem 4.3 give the pullback result. The unrolling construction terminates because T is a fixed natural number. $\square$

The construction is effective and may expand patches or unroll time. No optimal representation size is claimed. The artifact supplies valid stride-one convolution, one-head attention and representative grammar lowerings, rather than an exhaustive implementation of every stride, group, mask or architecture configuration in this theorem. It does not import arbitrary framework models.

6. Regularity, control flow, state and custom backward rules

Proposition 6.1 (stable-trace correctness). Suppose a terminating execution uses finitely many smooth operations and predicates. If an open neighborhood of the input executes the same finite instruction trace with the same discrete choices, the tape transformation of that trace computes the classical derivative of the program on that neighborhood.

Proof. On that neighborhood the program agrees with one finite smooth composition. Apply Theorem 4.3. $\square$

Termination at one input does not imply the neighborhood hypothesis. At a routing boundary, a recorded trace need not describe the local function. The artifact implements static finite graphs, not an interpreter for these dynamic traces.

Definition 6.2 (declared reverse rule). A selected or surrogate reverse rule has the type of K but carries an explicit rule label instead of proof (3). For ReLU, the artifact selects 1 at positive inputs and 0 at nonpositive inputs. On open sign regions this is the classical derivative; at zero it is a convention. A derivative proof on regions, a subgradient claim and a surrogate-gradient claim are different specifications. The same distinction applies to max ties, quantization and discrete routing.

An opaque operator must provide its forward denotation, residual interface, effects and an applicable reverse contract. A custom-VJP registration does not prove that the supplied rule is a derivative. The JAX interface explicitly allows separate forward/residual and backward definitions [6]. PyTorch’s derivative registry is an important inventory, with formulas, saved variables and differentiability information; a registry entry alone is not a proof for an arbitrary compiled kernel [7]. The reference model rejects opaque operations.

For a differentiable state transition $f : S \times X \rightarrow S \times Y$, both state coordinates enter the joint derivative. Stopping derivatives through a state coordinate is an explicit projection/convention. An optimizer transition using a computed gradient is an additional program. Hooks, mutation and external writes require a state/effect semantics beyond a pure tape contract.

For randomness, condition on a seed or explicit noise input when the conditional program is smooth. Interchanging differentiation and expectation needs separate integrability and domination hypotheses. No unbiased stochastic-gradient theorem follows just by differentiating one realization, and no sampler-gradient theorem is proved here.

7. Compiler and numerical execution contracts

Definition 7.1 (paired exact refinement). A backend or compiler pass refines a tape contract exactly when its representation relations preserve forward values, reachable residuals, cotangent zero/addition, and each local backward action. For exact encodings e_A, e_B and a tape relation ρ , this includes

$$\widehat{F}(e_A a) = (e_B f(a), \hat{r}), \quad \rho(r_a, \hat{r}), \quad \widehat{K}(\hat{r}, e_B^* u) = e_A^* K(r_a, u). \quad (8)$$

Relations must be stable under the product, projection and accumulation constructions used by the pass. Exact mathematical implementations satisfy this notion; general floating-point implementations need not. Restricting inputs to representable values does not ensure that every intermediate result is exactly representable.

Theorem 7.2 (conditional compiler preservation). A translation preserving types and graph wiring, with local paired exact refinements for all its operators and structural maps, preserves the whole forward and pullback denotations under its representation relations.

Proof. Induct on the forward instructions to relate values and tapes. Induct in reverse topological order to relate cotangents. At a shared argument, preservation of zero and addition relates the accumulated contributions. Apply (8) at each local step and the output projection. $\square$

This identifies proof obligations; it is not a verified lowering to PyTorch, JAX or StableHLO. StableHLO supplies an extensive operation/execution specification, but its custom calls are implementation-defined and floating-point reduction order can vary [5]. A backend claim must pin the relevant release, dtype, accumulation scheme, exception behavior and custom-call implementation.

Theorem 7.3 (conditional numerical error recurrence). Write the exact forward SSA state as $z_i = H_i(z_{i-1})$ and let $\hat{z}_i$ be a numerical state decoded to the same real coordinate space. Suppose, on a region containing the states and comparisons used below,

$$\|H_i(\hat{z}_{i-1}) - H_i(z_{i-1})\| \leq L_i \|\hat{z}_{i-1} - z_{i-1}\|, \quad \|\hat{z}_i - H_i(\hat{z}_{i-1})\| \leq \delta_i.$$

Let $K_i(z_i, -)$ be the exact pullback of H_i , reading its saved state. Assume uniformly on that region

$$\begin{aligned} \|K_i(\hat{z}_i, w) - K_i(\hat{z}_i, v)\| &\leq M_i \|w - v\|, \\ \|K_i(\hat{z}_i, v) - K_i(z_i, v)\| &\leq A_i \|\hat{z}_i - z_i\| \|v\|, \\ \|\widehat{K}_i(\hat{z}_i, \hat{v}_i) - K_i(\hat{z}_i, \hat{v}_i)\| &\leq \eta_i. \end{aligned}$$

Here K_i must be defined on those saved states, including the numerical ones. Numerical reverse updates include accumulation rounding. With $v_{i-1} = K_i(z_i, v_i)$, $\hat{v}_{i-1} = \widehat{K}_i(\hat{z}_i, \hat{v}_i)$, $e_i = \|\hat{z}_i - z_i\|$ and $d_i = \|\hat{v}_i - v_i\|$,

$$e_i \leq L_i e_{i-1} + \delta_i, \quad d_{i-1} \leq M_i d_i + A_i e_i \|v_i\| + \eta_i. \quad (9)$$

Proof. For the forward bound, insert $H_i(\hat{z}_{i-1})$ and apply the triangle inequality. For the reverse bound, insert $K_i(\hat{z}_i, \hat{v}_i)$ and $K_i(\hat{z}_i, v_i)$ between the numerical and exact update. Bound the three differences by η_i , $M_i d_i$ and $A_i e_i \|v_i\|$. $\square$

The output projection initializes v_n ; identical exact seed placement gives $d_n = 0$. Rounded input decoding may give $e_0 > 0$. Suitable A_i follow from bounded variation of the local derivative, for example bounded second derivatives on an appropriate neighborhood. Mere C^1 regularity does not guarantee such a Lipschitz bound. Establishing the constants, staying in the region and excluding

overflow are additional obligations. The artifact does not establish them and makes no hardware error-bound claim. The target is the derivative of the ideal real program, not a classical derivative of the discrete floating-point execution function.

Proposition 7.4 (exact functions versus observations). If $f = g$ as C^1 functions on an open domain, their pullbacks agree there. Equality after rounding or at one input need not imply pullback equality.

Proof. Equal functions have equal difference quotients, hence equal derivatives. For the second assertion, take $f(x) = 0$ and $g(x) = x^2$ on $(-1/2, 1/2)$. Nearest-integer rounding observes zero for both, but their derivatives are 0 and $2x$. At $x = 0$, $f(x) = x$ and $g(x) = 0$ give pointwise equal values and different derivatives. $\square$

Thus a claim that two correct smooth tapes for the *same exact function* have different mathematical pullbacks would contradict (3). Differences can persist in residual resources and execution effects; different observed losses can also conceal different underlying exact functions or declared backward rules.

8. Residual symmetries and a presheaf infinity-topos

Fix ambient spaces A, B and an open $U_0 \subseteq A$. Let $\mathcal{C}$ be the small poset of its open subsets, in a fixed set-theoretic universe, ordered by inclusion. For each U , let $\mathcal{I}(U)$ be the groupoid of the smooth tape contracts of Definition 4.1 for maps $U \rightarrow B$.

Definition 8.1 (residual re-encoding). An arrow between contracts $t = (f, R, F, K)$ and $t' = (f, R', F', K')$ is a diffeomorphism $h : R \rightarrow R'$ satisfying

$$F' = (\text{id}_B \times h)F, \quad K'(h(r), u) = K(r, u) \quad \text{for all } r, u. \quad (10)$$

These arrows compose and invert, yielding a groupoid. Proofs of (3) are treated as propositions, not an additional collection of nontrivial proof objects. We regard diffeomorphisms as a discrete set of arrows; no topology on their collection is imposed.

Restricting F along $V \subseteq U$ gives a presheaf of groupoids. Let $\mathcal{O}(U)$ be the discrete set of smooth functions $U \rightarrow B$. Taking nerves gives a map $q : \mathcal{I} \rightarrow \mathcal{O}$ in

$$\mathcal{X} = \text{Fun}(\mathcal{C}^{\text{op}}, \mathcal{S}),$$

the presheaf infinity-topos of spaces. The poset $\mathcal{C}$ is the indexing category; $\mathcal{X}$ is the semantic topos. No assertion that the groupoid-valued presheaf is already a sheaf for open covers is needed.

For nonempty U , every smooth $f : U \rightarrow B$ admits the mathematical tape $R = U$, $F(a) = (f(a), a)$, $K(a, u) = Df(a)^*u$. For empty U , use the terminal residual and zero backward map. Consequently q is objectwise surjective on connected components and is an effective epimorphism in $\mathcal{X}$. These mathematical tapes need not have executable finite syntax. The tensor compiler theorem concerns the executable subclass specified earlier.

Proposition 8.2 (nontrivial implementation loops). A forward-observation fiber of q can have nontrivial fundamental group, even for a constant forward function and a correct zero pullback.

Proof. Take $f = 0$, $R = \mathbb{R}$, $F(a) = (0, 0)$ and $K(r, u) = 0$. The nonidentity diffeomorphism $h(r) = -r$ preserves (10) and satisfies $h^2 = \text{id}$. It is a nontrivial automorphism of this tape. The fundamental group of the nerve of a groupoid at an object is its automorphism group. Since the forward-function target is discrete, this automorphism also defines a loop in the homotopy fiber. $\square$

More generally, append an unused residual coordinate saved as zero and allow compatible invertible re-encodings. The exact function and pullback do not determine such structure. This groupoid model is 1-truncated: its homotopy groups above degree one vanish. It retains implementation symmetries but does not, by itself, provide nontrivial higher homotopy groups in every degree. Higher certificate spaces would require a further construction.

9. Which policies descend to observations?

Definition 9.1 (policy). A policy on implementations is a subobject $P \hookrightarrow \mathcal{I}$ in $\mathcal{X}$. It is therefore compatible with restriction and residual isomorphism. It is observable through q if $P \simeq q^*Q$ for a subobject $Q \hookrightarrow \mathcal{O}$.

Theorem 9.2 (policy descent). Let $q : I \rightarrow O$ be any effective epimorphism in an infinity-topos. With projections $p_1, p_2 : I \times_O I \rightrightarrows I$, a policy $P \hookrightarrow I$ descends if and only if

$$p_1^*P \simeq p_2^*P \quad \text{as subobjects of } I \times_O I. \quad (11)$$

The descended subobject is unique up to unique equivalence.

Proof. Necessity follows by pullback. For sufficiency, classify P by a map $I \rightarrow \Omega$. Condition (11) identifies its two pullbacks to the kernel pair. Subobjects are propositions: their comparison, when it exists, is unique, so the descent coherences on the Čech nerve are automatic. Effective epimorphy identifies O with that nerve’s colimit. Mapping into Ω therefore gives the unique factorization $O \rightarrow \Omega$, which classifies Q . $\square$

This is standard effective descent applied to implementation policies [8]. In spaces it is invariance across homotopy fibers. In a general infinity-topos (11) is internal and concerns all generalized elements; checking ordinary global points alone is not sufficient. For a non-epimorphic observation map, replace its target by its image and apply the theorem to the resulting effective epimorphism.

Corollary 9.3 (resource observability obstruction). Exact forward-function observation does not determine a residual-dimension budget.

Proof. The constant map and zero pullback admit both a terminal residual and the one-dimensional residual of Proposition 8.2. The policy $\dim R = 0$ distinguishes them. It is invariant under diffeomorphism and restriction, hence defines a policy, but fails (11). $\square$

A policy about the exact pullback being Df^* is already true for every object of $\mathcal{I}$. Detecting a false custom backward rule requires a larger space of declared, potentially uncertified tapes and an observation that retains the relevant rule or certificate. Forward numerical equality cannot supply that missing proof.

10. Reference artifact and reproducibility

The version 1.0.0 artifact contains `tensor_ir.py`, `architectures.py`, `verify.py`, this manuscript, a typeset PDF, build instructions and machine-readable results. The reference backend is NumPy float64. It implements a shape-checked static SSA builder, explicit saved-value tapes, VJPs, separate JVPs, serialization and elimination of identity reshapes. This rewrite preserves real paired semantics. It can change the grouping of backward accumulation in floating point: an edge test obtains gradients 1 and 0 before and after removing an identity reshape from one shared branch with contributions $10^{16}, -10^{16}, 1$. Thus even deleting a structural identity needs a numerical contract if bitwise gradients matter.

Run `python verify.py --output verification-results.json` with Python 3.10 or later and NumPy. The recorded run used Python 3.12.14, NumPy 2.3.5 and seed 20261021. It required no network, GPU or external model framework.

Check	Recorded evidence
Independent rational polynomial oracle	512 scalar DAGs; 2,048 exact value/full-gradient evaluations; 1,987 nonzero gradients
Exhaustive small gather adjoints	All 494 index maps with input and output lengths from 1 through 4; repeated indices included
Finite set policy descent	249 surjections; 6,726 Boolean-policy cases
Smooth tensor programs	201 graphs; 603 directional and adjoint-pairing checks
Serialization and structural rewrite	201 round trips and 201 paired comparisons
Edge cases	10 checks, including repeated uses, selected ReLU, a rounding-sensitive rewrite, invalid shapes/indices, domain failures and opaque-call rejection

The polynomial oracle expands a multivariate coefficient representation and compares its exact derivatives with a separately evaluated rational reverse sweep. It does not execute the floating-point interpreter in exact arithmetic. Tensor checks compare the explicit reverse sweep with a JVP pairing and compare JVPs with central finite differences. The maximum normalized adjoint discrepancy was 4.30×10^{-16} and maximum relative directional discrepancy was 4.45×10^{-10} . The finite-difference step was 2×10^{-6} , with acceptance threshold 2×10^{-6} . These are deterministic finite checks, not universal error bounds.

Eight architecture examples were each run on five generated inputs: dense/tanh, residual, convolution, shared recurrence, attention, normalization, a tied DAG and parallel concatenation. Another 160 generated DAGs and one permutation/reduction/reciprocal example completed the numerical cases. The exhaustive set-level policy experiment does not validate infinity-topos descent by computation; Theorem 9.2 has a mathematical proof.

The implementation trusts its Python builder objects and assumes they are not mutated while a tape is in use. It is not a hostile-input sandbox. It rejects nonfinite intermediate values and explicitly checks reciprocal and inverse-square-root domains. Its elementary exponential softmax can overflow for large inputs. No dynamic loop engine, higher-order AD, proof-assistant formalization, hardware refinement, distributed semantics, general framework importer or comprehensive architecture test suite is supplied.

11. Consequences and next formal milestones

The operator question has a precise positive answer relative to $\mathcal{A}$: its programs lower to a finite smooth tensor inventory and possess correct compositional pullbacks. Broader coverage requires new constructors and proofs, rather than a change to the meaning of completeness. An implementation contract must also describe numerical and effect behavior.

A proof-assistant development can first encode the finite shape rules, exact scalar algebra and SSA induction, then formalize residual re-encodings and the local-to-global compiler theorem. A practical backend project can separately pin a framework release and discharge its primitive contracts, including nonsmooth conventions, reduction order and custom operators. The mathematical model permits these extensions without treating them as already completed.

Authorship, automation and evidence statement

The responsible submitting researcher is the anonymous author, who authorized preparation and submission. The byline intentionally omits personal names. An OpenAI assistant drafted the mathematics and reference code, checked cited primary-source pages, executed the listed checks and prepared the artifacts. Proofs are supplied in the text and were reviewed by the same assistant; there has been no independent referee validation or proof-assistant verification. Computational results describe the attached version only. No malicious payload or exploit implementation is included. All claims of derivative correctness refer to the explicit real-domain hypotheses; all numerical refinement claims are conditional as stated.

References

- [1] R. Cockett, G. Cruttwell, J. Gallagher, J.-S. P. Lemay, B. MacAdam, G. Plotkin and D. Pronk. **Reverse Derivative Categories**. CSL 2020, LIPIcs 152, 18:1-18:16. DOI: [10.4230/LIPIcs.CSL.2020.18](https://doi.org/10.4230/LIPIcs.CSL.2020.18).
- [2] M. Vakar and T. Smeding. **CHAD: Combinatory Homomorphic Automatic Differentiation**. ACM TOPLAS, 2022. DOI: [10.1145/3527634](https://doi.org/10.1145/3527634). [Open manuscript](#), [arXiv:2103.15776](https://arxiv.org/abs/2103.15776).
- [3] F. Lucatelli Nunes and M. Vakar. **CHAD for Expressive Total Languages**. Mathematical Structures in Computer Science 33(4-5), 311-426, 2023. DOI: [10.1017/S096012952300018X](https://doi.org/10.1017/S096012952300018X). [Open manuscript](#), [arXiv:2110.00446](https://arxiv.org/abs/2110.00446).
- [4] T. Smeding and M. Vakar. **Efficient CHAD**. 2023, [arXiv:2307.05738v3](https://arxiv.org/abs/2307.05738v3). [Primary manuscript](#).
- [5] OpenXLA. **StableHLO Specification**. Living official specification, consulted 21 October 2026. Sections on custom calls, reduction, dot algorithms and execution. [Specification](#). Cited as an operator/execution inventory; no release-independent backend theorem is inferred.
- [6] JAX authors. **Custom JVPs or VJPs with custom_jvp and custom_vjp**. Official documentation, consulted 21 October 2026. [Documentation](#).
- [7] PyTorch contributors. **Autograd derivative definitions, tools/autograd/derivatives.yaml**. Official source repository, main branch consulted 21 October 2026. [Source](#). A mutable inventory, not a pinned implementation certified here.
- [8] J. Lurie. **Higher Topos Theory**. Annals of Mathematics Studies 170, Princeton University Press, 2009; especially Chapter 6 and Section 6.2.3 on effective epimorphisms. [Author-hosted text](#).