

Forward and Pullback Operator Languages for Neural Networks: Completeness and Certification

Anonymous

21 October 2026

Abstract

We survey categorical and programming-language foundations for neural-network forward and reverse operators. We distinguish expressiveness, differentiation closure, derivative correctness, and implementation refinement. A residual interface and an architecture-relative theorem organize the results. We outline a homotopical certification program without claiming coverage of every implementation.

1. Purpose and status of the claims

Neural-network training combines operators, wiring, parameter sharing, reverse rules, numerical execution, and state. A complete forward/adjoint language must specify which of these structures it covers.

This survey connects reverse derivative categories [1], Combinatory Homomorphic Automatic Differentiation (CHAD) [2,3], circuit presentations [4], and tensor languages [8-11]. Sections 4-5 restate standard compositional reasoning; Sections 6-8 propose certification obligations and homotopical witness structures. No new AD algorithm or completed universal compiler is claimed.

Primary papers and official sources were checked on 21 October 2026. This selective synthesis is not a census of all architectures, framework extensions, or custom kernels.

Online documentation and implementation sources [8-12] were checked on 21 October 2026; their current content can change. The source bundle records URLs and the scope of each supporting claim.

2. Five meanings of completeness

Fix a typed operator signature Σ , a program language $\mathcal{L}_\Sigma$, a model class $\mathfrak{A}$, and a numerical execution configuration q .

Property	Required statement
Expressive coverage	Every model in $\mathfrak{A}$ has a semantics-preserving encoding in $\mathcal{L}_\Sigma$.
AD closure	The chosen target language can express every generated forward and reverse program.
Mathematical correctness	Generated pullbacks equal the dual derivatives of the denoted functions on the stated differentiability domains.
Equational completeness	The declared equations derive all semantic equalities for the specified model and equality notion.
Execution refinement	A backend implements the declared forward, reverse, and state behavior under q , exactly or within a specified contract.

These obligations are distinct. Finite functional completeness supplies no unique derivative; registries do not prove their rules; universal approximation supplies no exact program-translation theorem.

The phrase *adjoint* also needs a type. For a smooth map $f : U \rightarrow V$ between open subsets of Euclidean spaces, write

$$\text{jvp}_f(x, \dot{x}) = Df(x)\dot{x}, \quad \text{pb}_f(x, \bar{y}) = Df(x)^*\bar{y}.$$

Here $\bar{y} \in (\mathbb{R}^{\dim V})^*$ and the result belongs to $(\mathbb{R}^{\dim U})^*$. The dual is canonical; standard Euclidean pairings give its transpose representation. No categorical adjunction between nonlinear f and its reverse rule is asserted. Applying this to $f : \Theta \times X \rightarrow Y$ yields parameter and input cotangents.

3. Existing categorical and language foundations

Reverse derivative categories. Cockett and collaborators [1] axiomatize a reverse differential operator. Their analysis relates reverse differentiation to forward differentiation equipped with a compatible dagger on linear maps. This supplies architecture-independent equations for composition and accumulation, rather than a kernel inventory.

CHAD and expressive types. Vákár and Smeding [2] construct type-respecting, compositional forward and reverse transformations and prove correctness using logical relations. Lucatelli Nunes and Vákár [3] extend the method to expressive total languages with products, sums, inductive, coinductive, and function types. Under their hypotheses, primitive derivatives extend through language constructors. Arbitrary finite-precision execution remains a separate obligation.

Algebraic circuit completeness. Wilson and Zanasi [4] give generators and equations for polynomial circuits and analyze functional completeness over finite commutative semirings. Their comparator-based result establishes representability of arbitrary functions between finite powers of the carrier under the stated generators. Fullness need not imply faithfulness. Discrete reverse structure is additional algebraic data. IEEE floating-point arithmetic does not satisfy the theorem’s commutative-semiring hypothesis.

Efficiency and learning structure. Efficient CHAD [5] studies the complexity of reverse AD, including accumulation and array operations, and supplies an Agda formalization of its complexity

proof. A categorical account of gradient-based learning [6] combines parametrized maps, lenses, and reverse differentiation to organize models, losses, and optimizers. These address separate efficiency and learning-system obligations.

Architecture-level algebra. Gavranović and collaborators [7] propose a categorical deep-learning framework based on monads in a 2-category of parametrized maps. It connects architectural constraints and implementations. This position paper does not certify every existing implementation.

4. A typed forward/pullback operator interface

Let a tensor shape $s = (n_1, \dots, n_k)$ denote the finite-dimensional vector space

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

Products collect tensors; open refinements record domain conditions. Indices and shapes are static. Write A^* for the ambient vector-space dual.

Definition 4.1 (Residual contract). A primitive $g : A \rightarrow B$ has residual carrier R_g and functions

$$F_g : A \rightarrow B \times R_g, \quad B_g : R_g \times B^* \rightarrow A^*.$$

Its exact smooth contract requires g to be continuously differentiable and

$$F_g(a) = (g(a), r) \implies B_g(r, \bar{b}) = Dg(a)^* \bar{b}.$$

The result is real-linear in $\bar{b}$. Residuals save sufficient data; higher-order AD requires semantics for differentiating generated code. JAX custom VJPs [10] implement this forward/residual/backward pattern.

The following familiar rules describe a useful core. All indices are fixed, all sums are exact, and the matrix pairing is Frobenius.

Primitive	Pullback on output cotangents
Copy $x \mapsto (x, x)$	$(\bar{u}, \bar{v}) \mapsto \bar{u} + \bar{v}$
Discard $x \mapsto *$	Return zero input cotangent
Permutation or reshape P	Apply P^{-1}
Sum reduction	Broadcast to the reduced coordinates
Gather $y_i = x_{k_i}$	$\bar{x}_j = \sum_{i:k_i=j} \bar{y}_i$
Product $C = AB$	$(\bar{A}, \bar{B}) = (\bar{C}B^\top, A^\top \bar{C})$
Pointwise smooth φ	$\bar{x} = \varphi'(x) \odot \bar{y}$

Repeated gather indices accumulate cotangents; copy handles sharing. Named layers may be derived when both semantics are preserved.

Lemma 4.2 (Serial residual composition). Given contracts for $f : A \rightarrow B$ and $g : B \rightarrow C$, a contract for $g \circ f$ saves (r_f, r_g) and uses

$$B_{g \circ f}((r_f, r_g), \bar{c}) = B_f(r_f, B_g(r_g, \bar{c})).$$

Proof. The reverse chain rule gives

$$D(g \circ f)(a)^* \bar{c} = Df(a)^* Dg(f(a))^* \bar{c}.$$

Substitution of the two primitive contracts gives the claimed equation. $\square$

Theorem 4.3 (Compositional reverse correctness; standard). Consider a finite, well-typed acyclic graph built from primitives satisfying Definition 4.1, products, composition, copy, and discard. Suppose all domain obligations hold. Its reverse traversal, with cotangent accumulation at shared values, computes the pullback of its denotation.

Proof. Express the graph using the listed constructors. Primitive cases satisfy their contracts. Composition follows from Lemma 4.2; the derivative of a product map is block diagonal; dualizing copy adds the incoming cotangents; dualizing discard gives zero. Structural induction proves the claim. $\square$

The proof supplies no complexity bound: tree expansion can duplicate shared work. Efficient accumulation needs a separate cost argument [5].

Proposition 4.4 (Equations must preserve paired semantics). Let E be equations imposed on program syntax. A compositional interpretation into forward/pullback pairs factors through the quotient by E if and only if the two sides of every generating equation have equal paired interpretations.

Proof. Necessity follows by interpreting equal quotient classes. Conversely, equality of interpretations is a congruence, so it contains the congruence generated by E exactly when it contains every generator. The interpretation then descends. $\square$

For smooth real maps, equality as functions implies equality of derivatives. For specified finite-precision forward/backward pairs, equality of forward maps alone can fail this condition [13].

5. Architecture-relative completeness

Choose structural operators, constants, tensor arithmetic, finite reductions, fixed indexing, and scalar functions Φ on certified smooth domains. Contractions and convolutions derive from these, or remain named primitives with their own contracts.

Define $\mathfrak{A}_{\Sigma, N}$ to consist of the following well-typed constructions: primitive blocks from Σ ; serial and product composition; finite residual addition; parameter sharing by copy; and an explicitly bounded unrolling of at most N recurrent steps. Parameters are included among the inputs, so sharing accumulates their cotangents.

Theorem 5.1 (Relative coverage and differentiation). Assume a target language $\mathcal{L}$ has semantics-preserving encodings of the primitive contracts, the stated finite encodings of architecture constructors, and closure under the generated residual/reverse code. Then every architecture in $\mathfrak{A}_{\Sigma, N}$ has an encoding whose reverse code computes its mathematical pullback on its certified smooth domain.

Proof. Induct on the architecture construction. Primitive encodings provide the base. Composition and products translate their subterms. Residual addition uses copy followed by addition; parameter

sharing uses copy; bounded recurrence is finite composition with shared parameters. The resulting graph is finite and acyclic. Theorem 4.3 proves its reverse correctness; target-language closure supplies representability of that code. $\square$

Coverage is relative to these generators, constructors, domains, and closure assumptions. Instances include:

Construction	Additional obligation
Dense and convolutional blocks	Chosen smooth activation; finite shapes and index maps
Residual networks	Type-compatible addition and explicit parameter sharing
Attention	Dot products, exponentials, positive normalization denominators; fixed masks
Finite recurrent unrolling	A stated bound and accumulation for shared parameters
Fixed-graph message passing	Fixed adjacency and appropriate gather/reduction rules

For example, unmasked softmax attention is built from matrix products, scalar multiplication, exponentiation, division, and reduction. Numerically stable rewrites involving a maximum require their own piecewise and execution analysis. A ReLU network belongs to a piecewise-smooth extension, with a declared boundary rule; it is not globally covered by the continuously differentiable hypothesis above.

Boolean-circuit expressiveness supplies no classical derivative of a finite-bit machine function and cannot replace the differentiation assumptions.

6. From mathematical operators to actual training

Operator specifications. StableHLO [8] supplies a portable tensor-language specification, including program execution and individual operator semantics. PyTorch’s derivative registry [9] supplies formulas and differentiability annotations. JAX [10,11] exposes custom derivative rules and describes reverse AD through transposition of linearized computations. These support an inventory but do not replace implementation proofs. StableHLO’s implementation-defined custom calls require separately supplied meanings.

Nonsmooth and discrete behavior. ReLU boundaries, tied maxima, sorting, routing, and quantization require explicit treatment. An interface should distinguish a classical derivative, a selected generalized derivative, a declared surrogate, and a nondifferentiable result. Surrogate rules need not satisfy Definition 4.1 or the differential axioms.

State and randomness. A more complete forward interface has the form

$$F_g : A \times S \rightarrow B \times R_g \times S,$$

with an effect policy for reads, writes, failures, and random choices. The specification must identify differentiable and static state, permitted effects, and sample-path versus expectation derivatives. Interchanging differentiation and expectation requires hypotheses. Adaptive optimizers add their own transition state, as the learning framework [6] helps organize.

Finite precision. A configuration q must record dtype, device, compiler choices, accumulation order, and exceptional-value policy. Rounded execution F_g^q and its registered reverse code B_g^q need not be the forward map and derivative of one smooth real function. PyTorch’s numerical-accuracy note [12] documents limitations of cross-platform and algebraic reproducibility. The forward-extensionality obstruction in [13] explains why a loss-only contract is insufficient.

Commutativity and associativity of cotangent accumulation hold in the ideal algebraic semantics; floating-point accumulation needs an order or a bounded nondeterministic specification. Neither algebraic simplification over the reals nor a higher-precision rerun by itself proves equivalence of full training executions.

For a lowered program, the proposed obligation is a declared relation

$$\text{Sem}_q(\text{lower}(p)) \preceq_{\mathcal{K}} \text{Sem}(p),$$

where $\mathcal{K}$ specifies forward errors, reverse errors, effects, and protected state observations. This names an obligation, not an established uniform error theorem. Approximate composition needs stability bounds; security labels need margins or invariant proofs.

7. A homotopical organization of certification

The following is a proposed organization of certificates. Let $\mathcal{C}$ be a small category of typed contexts. Its morphisms must have specified restriction meanings, such as specialization of shapes or narrowing of admissible inputs. For each context, suppose implementations and certified reversible comparisons form a groupoid, compatibly under restriction. Then they define a presheaf

$$\mathcal{J} : \mathcal{C}^{\text{op}} \rightarrow \mathcal{S} \quad \text{in} \quad \mathcal{E} = \text{Fun}(\mathcal{C}^{\text{op}}, \mathcal{S}).$$

The presheaf category is an infinity-topos [14]. Groupoid examples are one-truncated; higher comparisons require more data. Sheaf descent additionally needs justified covers and gluing.

Let $U : \mathcal{J} \rightarrow \mathcal{V}$ forget reverse code, residuals, or execution certificates. At a reported forward behavior v , the homotopy fiber

$$\mathcal{J} \times_{\mathcal{V}}^h \{v\}$$

records implementations and witnesses of agreement with that report. Distinct components can retain different backward behavior; loops can retain implementation symmetries. A proposed compiler comparison must therefore preserve the chosen paired contract, not just lie over equal forward reports.

For a policy subobject $A \hookrightarrow \mathcal{J}$ and an effective-epimorphic observation $e : \mathcal{J} \rightarrow \mathcal{O}$, descent to observations requires

$$\text{pr}_1^* A = \text{pr}_2^* A \quad \text{on} \quad \mathcal{J} \times_{\mathcal{O}}^h \mathcal{J}.$$

This is the usual effective-descent condition [14], applied to policy classification in [13]. At the set level it means that observationally indistinguishable implementations have the same policy value. A certified operator inventory would make the relevant witnesses compositional; the survey does not construct all such witnesses.

8. A concrete next milestone

A credible first implementation would fix one pure tensor fragment, one numerical policy, and one back-end. Its deliverables would be a typed signature with primitive contracts, a forward/residual/reverse interpreter, graph-level reverse correctness, and explicit encodings of the architecture families in Section 5. It would report unsupported operations rather than silently treat them as certified.

Further milestones are stateful and stochastic extensions, checked kernel lowerings, coherent comparison certificates, and memory/cost semantics for checkpointing and recomputation. Conformance and adjoint-pairing tests find defects without proving universal correctness. The resource analysis in [5] is a useful separate foundation.

This paper supplies a source-and-claim ledger and manuscript build sources. It reports no new numerical experiment, framework coverage percentage, executable compiler, or proof-assistant verification. Its elementary proofs are included for clarity; the general categorical and AD results remain credited to their original sources.

Authorship and automation. The owner authorized anonymous preparation and submission. OpenAI ChatGPT/Codex assisted with literature retrieval, mathematical synthesis, drafting, and artifact checks. The present manuscript is submitted for editorial evaluation; no acceptance or independent referee report for it is asserted.

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. Vákár and T. Smeding. *CHAD: Combinatory Homomorphic Automatic Differentiation*. ACM TOPLAS 44(3), Article 20 (2022). [arXiv:2103.15776](https://arxiv.org/abs/2103.15776). The consulted online revision is v6 (2026).
3. F. Lucatelli Nunes and M. Vákár. *CHAD for Expressive Total Languages*. Mathematical Structures in Computer Science 33(4-5), 311-426 (2023). [arXiv:2110.00446v2](https://arxiv.org/abs/2110.00446v2).
4. P. Wilson and F. Zanasi. *An axiomatic approach to differentiation of polynomial circuits*. Journal of Logical and Algebraic Methods in Programming 135, 100892 (2023), Section 5. [Published text](#).
5. T. Smeding and M. Vákár. *Efficient CHAD*. [arXiv:2307.05738v3](https://arxiv.org/abs/2307.05738v3) (2023).
6. G. S. H. Cruttwell, B. Gavranović, N. Ghani, P. Wilson, and F. Zanasi. *Categorical Foundations of Gradient-Based Learning*. [arXiv:2103.01931v2](https://arxiv.org/abs/2103.01931v2) (2021).
7. B. Gavranović, P. Lessard, A. Dudzik, T. von Glehn, J. G. M. Araújo, and P. Veličković. *Position: Categorical Deep Learning is an Algebraic Theory of All Architectures*. [arXiv:2402.15332v2](https://arxiv.org/abs/2402.15332v2) (2024). Position paper.
8. OpenXLA contributors. *StableHLO Specification*. [Official specification](#).
9. PyTorch contributors. *Autograd derivative definitions*. [derivatives.yaml](https://pytorch.org/docs/master/autograd.html). Implementation source; main branch.
10. JAX contributors. *Custom derivative rules for JAX-transformable Python functions*. [Official documentation](#).
11. JAX contributors. *Primitives*. [Official documentation](#).
12. PyTorch contributors. *Numerical accuracy*. [Versioned documentation](#).
13. Anonymous. *Homotopy Fibers and Observation-Preserving Semantics for Finite-Precision Training*. Higher Categorical Semantics (2026), Editorially reviewed. [Canonical paper](#).
14. J. Lurie. *Higher Topos Theory*. Annals of Mathematics Studies 170, Princeton University Press (2009), Chapters 5-6. [Author's text](#).