Coherent Jet Gluing for Higher-Order Differentiation and Certified Distributed Optimization

Anonymous

Abstract. Huot, Staton, and Vákár prove correctness of forward automatic differentiation by gluing two diffeological interpretations along a relation on smooth plots. We replace the relation by a space or sheaf of witnesses and give an explicit exponential formula for the resulting comma ∞-category. Accessible left exact gluing over ∞-toposes yields an ∞-topos. An internal jet graph in smooth sheaves supplies the primitive witnesses, extends to function types and positive algebraic data, and commutes with changes of parameter base. The connected-plot specialization recovers the original diffeological relation and its first-order tangent theorem. Finite Weil algebras and quotient stacks supply higher jets and examples with nontrivial automorphisms. A Lean 4 implementation verifies lowering from an intrinsically typed higher-order calculus to stack code, proves transfer of jets from primitive laws, and checks an exact polynomial model. TCP workers execute the compiled code for directional derivatives and objective values. Across 25 small neural architecture instances and one higher-order structural case, the three-worker suite matched one-worker reductions exactly and decreased each measured objective over four accepted steps.

Keywords. Automatic differentiation, higher toposes, categorical gluing, Weil algebra, certified compilation, distributed optimization.

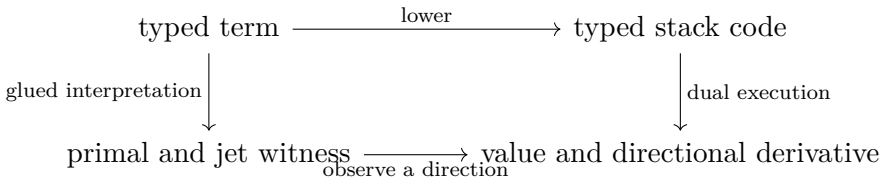
Contents

1 Problem and statements 2
2 Witness gluing in an ∞-category 3
3 Internal jets and recovery of the diffeological model 5
4 Finite Weil jets, continuations, and stack parameters 7
5 The Lean compiler 9
6 Distributing objective evaluations 11
7 Architecture examples and reproducible execution 14
8 Relationship to prior work and scope 18

1 Problem and statements

A differentiation compiler has three distinct interfaces. The source assigns a meaning to functions, closures, variants, and recursively shaped data. A translation assigns tangents to programs. The machine runs code at concrete inputs and distributes repeated evaluations. An equivalence at one interface does not imply an equivalence at the others. This separation is especially important for a higher-order term that returns a function, for a smooth parameter presented by a quotient stack, and for a distributed sum whose machine representation uses floating-point arithmetic.

Huot, Staton, and Vákár [1] pair a primal plot with its forward-AD plot in a relation on two diffeological spaces. They show that the gluing category supports the type constructors, and they recover the ordinary tangent map on ground types. Their later treatment gives Taylor translations for higher derivatives [2]. The contribution here is a relative ∞-categorical version in which witnesses, paths between witnesses, and parametrized families enter the same universal property. The construction has a concrete recovery functor to the original gluing. On the computational side, we implement a typed stack compiler and give proofs about the executable definitions, including a reduction theorem for exact arithmetic and a line-search theorem for an ordered score.



The lower comparison is a semantic equation under the primitive jet laws. When the base is $\mathbb{R}$, the supplied direction determines a tangent at the chosen point. The upper comparison is an equality in Lean for all typed terms, environments, stack continuations, and scalar interpretations.

We use the following numbering of results.

1. Theorems 2.2 and 2.4 construct the relevant operations in the higher comma category and give a criterion for it to be an ∞-topos.
2. Theorems 3.1 and 3.2 lift differentiable programs and give an exact recovery of the connected-plot model. Section 4 treats finite Weil jets, continuations, and a quotient-stack example.
3. Theorems 5.1, 5.2 and 6.1 connect the typed compiler to semantic jets, coordinate jobs, and finite-shard reduction. Section 7 reports numerical and network evidence separately.

We use *higher-order* for types with arrows and *higher categorical* for diagrams with non-discrete mapping spaces. The executable fragment covers simple types generated by reals, unit, the empty type, products, sums, arrows, and lists with structural fold. At the semantic level we also treat positive polynomial datatypes and arbitrary lawful smooth primitives. Executable optimization uses finite real parameter coordinates and scalar objectives. All the hypotheses on the real and machine models appear where the relevant results are stated.

2 Witness gluing in an ∞-category

Let $\mathcal{C}$ be an ∞-category with finite products and let $\mathcal{W}$ have pullbacks. Let $F\colon \mathcal{C} \to \mathcal{W}$ be a functor. In our applications $\mathcal{C} = \mathcal{M} \times \mathcal{M}$ separates the primal and translated semantics. We work in fixed universes in which the stated mapping spaces and colimits exist.

Definition 2.1 (Witness gluing)**.** Write

$$\mathcal{G}_F = \mathcal{C} \times_{\mathcal{W},\, F,\, \mathrm{ev}_1} \mathrm{Fun}(\Delta^1, \mathcal{W}).$$

An object is $(A, R, \rho\colon R \to FA)$. A morphism to (B, S, σ) consists of $f\colon A \to B$, $u\colon R \to S$, and a specified path $Ff\,\rho \simeq \sigma u$. The paths and their higher compatibilities are part of the mapping space.

In particular,

$$\mathrm{Map}_{\mathcal{G}_F}((A,R,\rho),(B,S,\sigma)) \simeq \mathrm{Map}_{\mathcal{C}}(A,B) \times_{\mathrm{Map}_{\mathcal{W}}(R,FB)} \mathrm{Map}_{\mathcal{W}}(R,S). \tag{1}$$

The two legs send f to $Ff\,\rho$ and u to σu. This formula follows from the space of squares in the arrow category and gives the interpretation of the pullback in Definition 2.1. For example, when $\mathcal{C} = \mathcal{W} = \mathcal{S}$ and $F = \mathrm{id}$, the mapping space from $(*, *, \mathrm{id})$ to $(*, BG, !)$ is BG for a discrete group G. Its loops are genuine gluing data. The monomorphism restriction used for ordinary binary relations removes this example.

Theorem 2.2 (Operations in the gluing)**.** *Suppose $\mathcal{C}$ and $\mathcal{W}$ are cartesian closed and F preserves finite products. Finite products and exponentials exist in $\mathcal{G}_F$ and the projection to $\mathcal{C}$ preserves them. Colimits of shape I exist and are computed on bases and witnesses whenever both categories admit them. If both categories admit countable coproducts, list objects exist and the projection preserves them.*

Proof. The product of (A, R, ρ) and (B, S, σ) has base $A \times B$ and witness $R \times S$, with its map to $F(A \times B)$ given by product preservation. The terminal object is $(1, 1, !)$. Let $\alpha\colon F(B^A) \to (FB)^R$ be adjoint to

$$F(B^A) \times R \longrightarrow F(B^A) \times FA \simeq F(B^A \times A) \xrightarrow{F(\mathrm{ev})} FB.$$

The exponential has base B^A and witness object

$$E = F(B^A) \times_{(FB)^R} S^R, \tag{2}$$

with first projection as its structure arrow. The second leg of the pullback is σ^R. For a test object (C, T, η), a point of the mapping space into this exponential is a map $C \to B^A$, a map $T \to S^R$, and a homotopy in $\mathrm{Map}_{\mathcal{W}}(T, (FB)^R)$ between the composites fixed by η, α, and σ^R. Currying in $\mathcal{C}$ and in $\mathcal{W}$ identifies the entire mapping space with the space in (1) for $(C \times A, T \times R)$ and (B, S). The equivalence is natural in the test object.

For a diagram $(A_i, R_i, \rho_i)_{i \in I}$, the maps $R_i \to FA_i \to F(\mathrm{colim}_i\, A_i)$ assemble to $\mathrm{colim}_i\, R_i \to F(\mathrm{colim}_i\, A_i)$. Mapping out of the two colimits gives a limit of the pullbacks in (1). This proves the colimit formula without a colimit preservation hypothesis on F. In any cartesian closed

category with the stated coproducts, $\coprod_{n\geq 0} X^n$ is a list object. The empty list is the zeroth summand and cons is the shift induced by distributivity. For an algebra (z, s) for $1 + X \times -$, the map out of the nth summand is the n-fold composite of s with base z. With homotopies included, the space of such algebra maps is contractible. Indeed, fixing the previous components makes the next map together with its path to the prescribed composite a contractible space. The limit over n gives the claimed mapping space. The projection preserves the coproduct and each finite power, so it preserves lists and their folds. □

The witness part of (2) is a function compatibility condition, parametrized by R. It does not rely on preservation of exponentials by F. The distinction matters for plot functors and exponentiation by the curve object.

Proposition 2.3 (Finite-branching polynomial datatypes)**.** *Under the hypotheses of Theorem 2.2, assume countable coproducts. For a finite family of coefficients B_i and natural numbers d_i, the polynomial $P(X) = \coprod_i B_i \times X^{d_i}$ has an initial algebra in $\mathcal{G}_F$. The projection to $\mathcal{C}$ preserves it.*

Proof. Let $\mathcal{T}$ be the countable set of finite ordered rooted trees whose nodes labelled by i have d_i children. Associate to a shape t the object $B_t = \prod_{v \in t} B_{\ell(v)}$ and take $L = \coprod_{t \in \mathcal{T}} B_t$. Distributivity of finite products over coproducts identifies $P(L)$ with L by grafting at a root. For an algebra $P(X) \to X$, a map out of L consists of its maps on summands. The algebra-map condition on a summand says that its map, with a specified homotopy, equals the result of applying the target algebra to the already specified child maps. At each finite height these conditions are products of contractible extensions over the lower heights. Their limit is contractible. If there are no finite trees the initial object is the required algebra. The entire construction uses operations preserved by projection. This is the finite-branching form of the usual polynomial and W-type construction [8]. □

Theorem 2.4 (Artin gluing criterion)**.** *If $\mathcal{C}$ and $\mathcal{W}$ are ∞-toposes and F is accessible and left exact, $\mathcal{G}_F$ is an ∞-topos.*

Proof. The comma construction is accessible by accessibility of the three categories and of F and the evaluation functors. It admits all small colimits by the colimit formula, so it is presentable. Finite limits are computed on bases and witnesses because F is left exact. The two projections jointly detect equivalences. The ∞-Giraud conditions now hold componentwise. Universality of colimits follows by pulling back in each topos. Disjointness follows by computing the pullback of two distinct coproduct injections in each topos. A groupoid object and its realization project to groupoid objects and realizations. Comparing to the Čech nerve of the augmentation gives equivalences on both projections by effectiveness there, and hence an equivalence in $\mathcal{G}_F$. We use the presentable version of the ∞-Giraud characterization in [4, §6.1]. For gluing and recollements from a stratified perspective, see [5]. □

Proposition 2.5 (Change of gluing data)**.** *A commuting square $F'U \simeq VF$ induces a functor $\mathcal{G}_F \to \mathcal{G}_{F'}$ by applying U to bases and V to witnesses. If U and V preserve finite products, pullbacks, exponentials, and the coproducts used in the type theory, it preserves the corresponding constructors and their structure maps.*

Proof. The comparison at $F'U$ transports the structure arrow of each object. On morphisms it transports the homotopies in (1). Functoriality of the arrow category supplies all coherences. The assertions about products, colimits, and exponentials follow by applying the preservation hypotheses to their formulas, in particular to the pullback (2). □

3 Internal jets and recovery of the diffeological model

Let $\mathcal{E} = \mathrm{Sh}_\infty(\mathrm{CartSp})$ be the ∞-topos of sheaves of spaces on the smooth Cartesian site. The site may use open Euclidean sets as chart objects, with ordinary open covers. Write L for the sheaf represented by the smooth real line and take $K = L$. Products of chart objects represent products of sheaves. For any chart U, the value of L^K on U is the discrete space of smooth maps $U \times \mathbb{R} \to \mathbb{R}$. Define the morphism

$$\mathsf{j}\colon L^K \longrightarrow (L \times L)^K, \qquad f(u,t) \longmapsto (f(u,t), \partial_t f(u,t)). \tag{3}$$

Smoothness in both variables and the chain rule for pullback along maps in U give a natural transformation of sheaves. The variable t is distinguished and held fixed by such changes of U. We do not demand that (3) commute with arbitrary reparametrizations of t while keeping its tangent component unchanged. Under $t = h(s)$ the second component is multiplied by $h'(s)$.

Use $F(A_0, A_1) = A_0^K \times A_1^K$ on $\mathcal{C} = \mathcal{E} \times \mathcal{E}$, with $\mathcal{W} = \mathcal{E}$. This functor is accessible and left exact. Exponentiation by K is a right adjoint between presentable categories and the product is a finite limit. The glued scalar object is

$$\mathbf{L} = ((L, L \times L), , L^K, \langle \mathrm{id}, \mathsf{j} \rangle).$$

For each n-ary smooth primitive $p\colon L^n \to L$, equip its primal interpretation with

$$p_{\mathsf{D}}((x_i, v_i)_{i=1}^n) = \left(p(x), \sum_i \partial_i p(x) v_i \right). \tag{4}$$

Constants have zero second component. The map of witness objects is postcomposition by p on K-plots. Compatibility of this map with the two coordinates is exactly the ordinary smooth chain rule on $U \times \mathbb{R}$. For primitives defined on an open domain the formula is used on plots taking values in that domain. The total typed semantics assumes globally smooth primitives. A locally defined operation gives the same result for an evaluation whose intermediate calls remain in a neighborhood on which those formulas are smooth.

Theorem 3.1 (Coherent jet lifting)**.** *A smooth primitive signature with the lifts* (4) *interprets the simply typed language with finite products, sums, functions and lists in $\mathcal{G}_F$. The base projections are the primal semantics and the dual-number translation. Every well-typed term has a witness map over this pair. The lifting also interprets the datatypes of Proposition 2.3. The construction is compatible with reindexing of a parameter base B in $\mathcal{E}$.*

Proof. Interpret the ground generator as $\mathbf{L}$, and primitive generators as the maps just constructed. For a context use the finite product. Variables are projections. Substitution is composition.

Pairing and elimination use products, variants use the coproduct structure, abstraction and application use the exponential adjunction of Theorem 2.2, and fold uses initiality. The two projections preserve these operations and so act on terms by the stated primal and translated clauses. More explicitly, the arrow type translates to $\mathsf{D}(A \to B) = \mathsf{D}(A) \to \mathsf{D}(B)$ because the base of the glued exponential is the pair of exponentials. The witness part expresses preservation of related arguments over each generalized parameter. The equations of the calculus are induced by the universal properties. For example, both sides of a fold substitution correspond to the same algebra map, whose space of choices is contractible. Functoriality of composition and of the universal operations supplies compatibility for diagrams of substitutions. This construction requires no choice of an ordinary tangent bundle on a function type.

For the relative assertion, do the same construction in $\mathcal{E}_{/B}$, using $L_B = B \times L \to B$ and $K_B = B \times K \to B$. For $f\colon B' \to B$, pullback preserves finite limits and colimits in the slice. It also preserves slice exponentials. To see this, apply the adjunctions for pullback and dependent sum to the functor represented by the slice exponential, and use the pullback formula $\Sigma_f(Z \times_{B'} f^*A) \simeq \Sigma_f Z \times_B A$. The primitive graph is the pullback of (3), hence it commutes with reindexing. Apply Proposition 2.5 to the two slice gluing data. The comparison equivalences compose because pullback and the universal constructions are functorial. □

3.1 Monomorphism witnesses and connected plots

For an ordinary diffeological space X, let $P_U X = \mathbf{Diff}(U, X)$ for a nonempty connected chart U. The ordinary category **Diff** is cartesian closed and cocomplete, and the usual embedding of manifolds preserves the relevant finite products and countable disjoint sums [7, 1]. Take

$$\mathcal{C} = N(\mathbf{Diff} \times \mathbf{Diff}), \qquad \mathcal{W} = \mathcal{S}, \qquad F(X, Y) = (P_U X \times P_U Y)_{\mathrm{disc}}.$$

Consider the full subcategory whose structure maps are monomorphisms. A monomorphism to a discrete space is equivalent to the inclusion of a subset. The mapping spaces of this subcategory are discrete by (1). A morphism is uniquely specified by a pair of smooth maps that preserves the relation. The original category in [1, Proposition 1] is exactly this model, with $U = \mathbb{R}$ for the derivative relation.

Theorem 3.2 (Recovery and tangent comparison)**.** *The monomorphism-witness subcategory for the connected plot functor above inherits products, exponentials, finite sums, and lists. With $U = \mathbb{R}$ and scalar graph $c \mapsto (c, c')$, its induced interpretation agrees with the forward gluing of [1]. For first-order contexts and outputs generated by real, unit, products, sums and lists, its dual map is the ordinary tangent bundle map under the canonical identifications of the ground types.*

Proof. The product of two monomorphisms is monic. In (2), $S^R \to (FB)^R$ is monic and remains so under pullback. A plot from connected nonempty U into a diffeological disjoint sum has a single tag. The map on a witness coproduct takes the same tag in both coordinates. Distinct tags have disjoint images and each summand is injective. Thus the inherited coproduct is monic. This holds also for the countable coproduct of powers used for lists. The scalar object is the graph of the derivative on curves. Primitive arrows are the dual-number arrows and the operations agree by the formulas of Theorem 2.2. This proves equality of the gluing interpretations up to their canonical structure isomorphisms.

Each first-order type is a countable disjoint union of Euclidean spaces of locally finite dimension, by distributivity and the description of lists as a coproduct of powers. Its transformed interpretation is its tangent bundle by product and disjoint-sum preservation of the ordinary tangent functor. Let g be the primal of a term and g_{D} its dual. The witness property for a curve c gives $g_{\mathsf{D}}(\dot{c}(t)) = (g \circ c)^{\cdot}(t)$, where the dot denotes the tangent lift with its base point. Given a point and tangent in a component, a curve realizing the tangent exists in that component. Evaluating at that point yields $g_{\mathsf{D}}(x, v) = (g(x), Dg_x v)$ on all tangent vectors. This gives Theorems 1 and 2 of [1], including terms that build and apply functions internally. □

3.2 Descent of a parametrized interpretation

Corollary 3.3 (Descent)**.** *Let $U \to B$ be an effective epimorphism in $\mathcal{E}$ with Čech nerve $U_\bullet$. The gluing categories of the relative construction satisfy*

$$\mathcal{G}_{F_B} \simeq \lim_{[n]\in\Delta} \mathcal{G}_{F_{U_n}}.$$

A coherent family of local jet witnesses descends together with its primal and translated arrows.

Proof. Descent for objects and morphisms in an ∞-topos identifies $\mathcal{E}_{/B}$ with $\lim_\Delta \mathcal{E}_{/U_n}$ [4, §6.1]. The reindexing equivalences in Theorem 3.1 identify the functor F_B with the limit of the relative plot functors. Apply the fact that limits commute with limits to Definition 2.1. The arrow-category functor preserves limits, since it is right adjoint to product with Δ^1 in $\mathbf{Cat}_\infty$. Finally apply the equivalence to the space of arrows with the given source and target. This statement uses the full coherent descent datum, including its higher paths. □

4 Finite Weil jets, continuations, and stack parameters

4.1 Jets of arbitrary finite order

Let A be a finite-dimensional commutative local $\mathbb{R}$-algebra with augmentation $A \to \mathbb{R}$ and nilpotent kernel $\mathfrak{m}$. Suppose $\mathfrak{m}^{r+1} = 0$. For a smooth function $p\colon \mathbb{R}^n \to \mathbb{R}$ and $x_i + u_i \in \mathbb{R} \oplus \mathfrak{m}$ define

$$p^A(x + u) = \sum_{|\alpha|\le r} \frac{\partial^\alpha p(x)}{\alpha!} u^\alpha. \tag{5}$$

The expression is independent of increasing the bound r past the nilpotence degree. It is natural under morphisms of augmented real algebras. For composable smooth maps, $(q \circ p)^A = q^A \circ p^A$. Indeed, the Taylor jet of the composite is obtained by substitution in the truncated polynomial algebra, with all terms of total degree greater than r in the ideal that vanishes upon evaluation at u. Evaluating the substitution identity in A gives the formula. The chain rule is its square-zero case.

For the free truncated algebra $A_{r,d} = \mathbb{R}[\epsilon_1, \dots, \epsilon_d]/(\epsilon)^{r+1}$, use $K_d = L^d$ where the exponent d here denotes a finite product. The graph sends a smooth family $f(u, t)$ with $t \in \mathbb{R}^d$ to the family whose coefficient of ϵ^α is $\partial_t^\alpha f(u, t)/\alpha!$. Let $L_{A_{r,d}}$ be the smooth sheaf represented by the

underlying finite-dimensional vector space of this algebra. This graph is a morphism into $L_{A_{r,d}}^{K_d}$ and (5) proves compatibility with the primitives. A choice of nilpotent generators of any finite Weil algebra gives a quotient $A_{r,d} \to A$ and the corresponding graph by postcomposition. The induced interpretations are natural for these quotient maps on the scalar graph. Every finite family of specified coefficients over a smooth point is realized by a polynomial plot in these probe variables, which makes the graph an observation test for ground Euclidean jets. The same formulas on chart overlaps define Weil prolongations of manifolds. Covariance of the prolonged transition maps follows from the substitution identity.

Corollary 4.1 (Parametric higher jets)**.** *Under the smooth primitive laws for* (5)*, the glued lifting interprets each well-typed term at any finite Taylor order, also for higher-order terms and finite-branching polynomial datatypes. At first-order types the interpretation agrees with the corresponding Weil prolongation, where the spaces are disjoint unions of Euclidean components.*

Proof. Replace **L** by the graph of the appropriate jet morphism. The product, exponential and algebra constructions are unchanged. At ground types, products and component tags determine the prolongation. A polynomial probe with the given jet in every coordinate reduces the comparison to the scalar substitution identity. The ground codomain's tag is locally constant along the probe, and the statement then holds in that component. This is compatible with the familiar higher-order Taylor translation [2]. It uses finite Weil algebras in the smooth topos. The general theory of tangent ∞-categories, including Goodwillie-calculus examples, has a different set of axioms and applications [6]. □

For instance, for $p(x) = x^3$, the second-order jet at x along $x + v\epsilon + a\epsilon^2$ is

$$x^3 + 3x^2 v\epsilon + (3x^2 a + 3xv^2)\epsilon^2.$$

The coefficient a is half the second derivative of a curve when the curve has that Taylor expansion. The extra $3x^2 a$ term shows why a second-order program must distinguish the jet of an input curve from the Hessian of the output function at a point. The executable in Section 5 implements first-order dual numbers.

4.2 A continuation comparison

Take the square-zero algebra $\mathbb{R} \oplus V$ with $V = \mathbb{R}^k$. The well-formed scalar continuation representation of $v \in V$ is $c_v(z) = zv$. Evaluation at 1 inverts this encoding on the linear subspace of $V^{\mathbb{R}}$. For a primitive p, the continuation rule sends the input continuations to

$$c(z) = \sum_i c_{v_i}(\partial_i p(x) z) = z \sum_i \partial_i p(x) v_i.$$

Thus it preserves the graph relating continuation representations to simultaneous tangent coordinates. Gluing the two scalar representations and using Theorem 2.2 lifts this comparison to the entire type grammar. At arrow types this is the exponential witness relation, which compares actions on related arguments. It does not assert that all functions on the two representations are equivalent. At first-order types, use the two ground encodings componentwise. Seed with e_i and

evaluate the resulting continuation at 1 to obtain the vector of partial derivatives. This recovers the comparison in Theorem 3 of [1]. The argument concerns the semantics of continuation values. An efficient reverse pass requires an additional representation and cost analysis of sharing, as in [3].

4.3 A quotient stack with nontrivial witnesses

Example 4.2 (Rotations of a plane)**.** Let the discrete cyclic group C_m, $m \geq 2$, act on $\mathbb{R}^2$ by rotations. Take the parameter object $X = [\mathbb{R}^2/C_m]$ in $\mathcal{E}$, and let $T_\Gamma X = [T\mathbb{R}^2/C_m]$ with the diagonal action on position and velocity. The squared norm descends to $g\colon X \to L$. Its lifted map is

$$g_{\mathsf{D}}\colon T_\Gamma X \longrightarrow L \times L, \qquad [x, v] \longmapsto (\|x\|^2, 2\langle x, v\rangle).$$

For a chart U, a generalized curve in X is a principal C_m-bundle $P \to U \times \mathbb{R}$ with an equivariant map $a\colon P \to \mathbb{R}^2$. The bundle projection is locally a diffeomorphism. The distinguished vector ∂_t has a unique lift along it. Sending a to $(a, da(\widetilde{\partial}_t))$ defines a functor on these bundle groupoids, compatible with change of U. Consequently it defines a jet arrow $X^K \to (T_\Gamma X)^K$. Equivariance and the scalar chain rule give the glued morphism over (g, g_{D}).

At the constant zero curve the automorphism group contains C_m. Its elements act as gauge transformations of the bundle that preserve the constant map. The jet functor carries them to the same automorphisms at the zero position and zero velocity. Passing to the orbit set erases these arrows. The Euclidean gradient $2x$ is equivariant, and an update $x \mapsto (1 - 2\alpha)x$ also descends. This example works with an étale discrete quotient and an invariant Euclidean metric. The result does not require that tangent prolongation preserve every colimit of smooth stacks. A choice of metric is essential for converting a differential covector to a gradient on a general parameter manifold.

5 The Lean compiler

5.1 Source and target

CertJet is a standalone Lean 4 package using the standard library. Its type grammar is

$$A, B ::= \mathrm{real} \mid 1 \mid 0 \mid A \times B \mid A + B \mid A \to B \mid \mathrm{List}(A).$$

The source terms include de Bruijn variables, signed decimal literals, real primitives, product formation and projection, empty elimination, abstraction and application, let, injections and case analysis, nil and cons, and right fold. List length and sum tags can be dynamic. Function values can be captured, passed, returned, and stored in algebraic data within a program. At the wire boundary a closure is used by composing it with a term in the same language. Values on the wire are first-order data.

Let $\mathsf{Val}_R(A)$ be the recursive interpretation of types over scalar carrier R. An environment is a heterogeneous product of such values. A scalar model chooses literal, unary, and binary

interpretations. The source evaluator eval_m is structurally recursive. The stack machine is indexed by context, input stack and output stack:

$$\mathsf{Code}\ \Gamma\ S\ T.$$

An instruction can only consume the declared top operands and pass the resulting stack to a well-typed continuation. For example, a binary opcode consumes two real operands and pushes a real result. A closure instruction stores a nested block with one new environment variable. A branch instruction stores two blocks whose input environment types can differ and whose output types agree. A fold instruction stores a block with two additional environment positions, one for an element and one for its accumulator. The interpreter run_m evaluates these typed blocks without type tags on the values in the Lean model.

The lowering function compiles an expression against a typed continuation. If $e\colon \mathsf{Term}\,\Gamma\,A$ and $k\colon \mathsf{Code}\,\Gamma\,(A :: S)\,T$, then $\mathsf{lower}(e,k)\colon \mathsf{Code}\,\Gamma\,S\,T$. The left operand of a binary operation is lowered first, then the right, and then the binary instruction. An abstraction is lowered to a closure containing a recursively compiled body. The operand of a case is lowered to a branch with two recursively compiled alternatives. The executable artifact is $\mathsf{compile}(e) = \mathsf{lower}(e, \mathsf{done})$.

Theorem 5.1 (Compiler preservation)**.** *For every scalar model m, typed term e, environment ρ, stack s, and well-typed continuation k,*

$$\mathsf{run}_m(\mathsf{lower}(e,k),\rho,s) = \mathsf{run}_m(k,\rho,\mathsf{eval}_m(e,\rho) :: s). \tag{6}$$

Consequently the first result of executing $\mathsf{compile}(e)$ *from an empty stack equals* $\mathsf{eval}_m(e,\rho)$*.*

Proof. Induct on the intrinsically typed source term, generalizing both stack shapes and the continuation. In the binary case, the hypotheses for the two operands reduce the run to the continuation applied to their two source values, in stack order. The opcode equation then gives the source binary result. In the abstraction case, the hypotheses for the body yield pointwise equality of the compiled closure with the source function in the extended environment. In the branch case, split on the semantic tag and apply the matching alternative hypothesis. In a fold, the step hypothesis applies to every extended environment, hence to each element and accumulator passed to the same list recursor. The empty-type eliminator discharges the unreachable branch. The remaining constructors follow their defining equations. The proof is checked in Lean as `lower_correct`, with the empty-stack consequence `compile_correct`. □

The CLI parses a JSON source tree and infers an intrinsically typed term. It emits a structured stack artifact containing context and result type. Blocks use a separate vocabulary of stack instructions, explicit nested code, and exact mantissa-and-scale literal pairs. At execution, the worker checks the block's stack transitions and constructs a value of the same dependent Code type before running it. Theorem 5.1 is a theorem about typed source and typed instructions. The parser, bytecode serializer and network framing are executable boundary code. Their wire behavior is tested in Section 7.

5.2 The jet transfer theorem

For scalar carriers C and R, let $j\colon C \to R \times R$ be a proposed jet map. A `Calculus` supplies a model on R and a local JVP for each primitive. The corresponding dual model acts on $R \times R$ by returning the value and that JVP. The primitive law requires j to preserve each literal and primitive after this translation. For example, for multiplication it requires

$$j(fg) = (xy, x\,\dot{y} + \dot{x}\,y) \quad \text{when } j(f) = (x, \dot{x}) \text{ and } j(g) = (y, \dot{y}).$$

Lift the scalar graph to types. At products use componentwise relation, at sums require the same tag, at lists use elementwise matching, and at arrows require that the two functions take related arguments to related results. Extend the relation to environments componentwise.

Theorem 5.2 (Lean jet transfer)**.** *For any two scalar models and scalar relation preserved by their literals and primitive operations, source evaluation at related environments yields related results at every type. In particular, under* `JetLaws` *for j, compiled dual execution at an environment related to a curve environment returns a value related to the primal curve result. At scalar output it is exactly the value of j on that result.*

Proof. Induct on the source term. A variable uses the related environment. Literals and primitives use their local laws. For abstraction, fix related arguments and use the hypothesis on the body in the extended environments. Application uses that function relation. Matching sums have the same tag, so the branch hypotheses apply to related payloads. For a fold, induct on the elementwise list relation. Apply the body hypothesis to the current pair of related elements and the pair of related accumulator values. The other cases are immediate from the structural relation. Apply (6) to the dual model. The Lean names are `fundamental`, `compiled_jet_sound`, and `compiled_scalar_jet`. □

For an analytic instance, take C to be a ring of smooth real curve germs at zero, with pointwise primitive interpretations on their domains. Take $j(c) = (c(0), c'(0))$. The elementary chain rule verifies the primitive laws. For higher-order types the Lean theorem uses the logical relation on the recursively interpreted values. The smooth-plot semantics of Theorem 3.1 independently supplies the geometric formulation for function types. For a fully checked exact instance, `Exact.lean` uses formal univariate integer polynomials, independently computes coefficients by Cauchy convolution, and proves that projection to the constant and linear coefficients satisfies the jet laws. The exact fragment uses integer literals and polynomial arithmetic. A closed computation with a captured scalar x compiles $(x+3)^2$ and verifies $(25, 10)$ at $(x, \dot{x}) = (2, 1)$ by definitional equality. The analytic laws for transcendental library functions are the mathematical hypotheses of the generic theorem. The kernel audit and numerical tests are reported separately.

6 Distributing objective evaluations

6.1 From any scalar objective to jobs

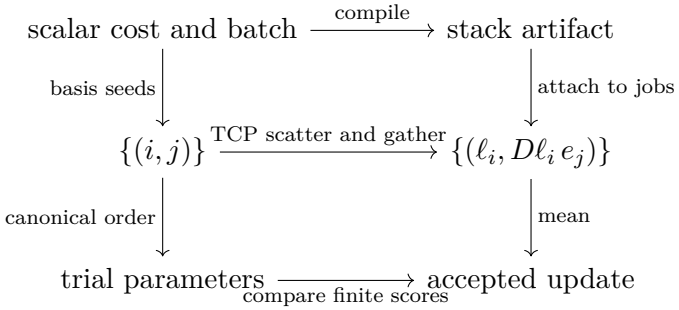
An arbitrary typed expression can be compiled as a value at its declared type. To optimize it, choose a finite real parameter chart and supply a scalar cost. A higher-order expression e can be

used inside an arbitrary well-typed cost expression $J(\theta, e)$ before lowering. The CLI accepts an optional cost body with the result of the original expression in its innermost binder and inserts that let before type checking. A network can use products or lists for tensors and folds or bounded unrolling for repeated layers. Neither a closed form for the loss nor an additive decomposition inside the program is required. For a sample s_i, write the resulting loss as $\ell_i(\theta)$. For a general coupled objective, give the whole input as one sample. For empirical risk, give a batch of n samples with identical parameter values and define

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(\theta), \qquad n > 0.$$

To optimize values nested in products, lists, or fixed sum branches, select real leaf paths. With the other leaves held fixed, the paths define a finite-dimensional affine family of environments. Branch tags and list sizes stay fixed within each seeded environment.

For each pair (i, j) with i a sample and j a selected coordinate, send the compiled bytecode and inputs with the basis direction e_j. Both primal values and JVPs return from the worker. The coordinator checks that the primal of a sample agrees across directions, restores the canonical input order, and reduces each coordinate. A single fully coupled loss still admits one job per coordinate. A sum of independent losses additionally admits data sharding.



Theorem 6.1 (Exact dispatch and reduction)**.** *Suppose the objective is smooth on a neighborhood of the evaluation point, all worker models implement the same lawful primitives, and every required job is returned once. With exact arithmetic, the assembled result is* $(L(\theta), \nabla L(\theta))$*. For any partition of the indexed jobs of a coordinate into finite shards, reducing each shard and reducing the partial sums yields the same result as the centralized sum.*

Proof. For the job (i, j) take the curve $t \mapsto \theta + te_j$ with all unselected inputs constant. By Theorem 5.2 the returned jet has second coordinate $D\ell_i(\theta)e_j$. Linearity of a finite sum gives $DL(\theta)e_j$ after reduction and division by n. The exact partition claim follows from associativity, the identity law, and commutativity of addition. More explicitly, a fold over a concatenation is the combination of its two folds. Induct on the list of shards, then use invariance under a list permutation to identify the flattened jobs with the canonical list. These are the Lean lemmas `total_append`, `total_shards`, `total_perm`, and `total_schedule`. □

The finite-index form of this result is the identity

$$\sum_{j \in J} \sum_{i \in f^{-1}(j)} x_i = \sum_{i \in I} x_i \qquad (f \colon I \to J).$$

For a commutative monoid object M in a cartesian ∞-category, the corresponding summation operations come from the commutative algebra structure in the cartesian monoidal category. A partition map decomposes the associated finite pointed-set operation into its fiber operations followed by the total operation. Functoriality gives the compatibility path, and composition of partition maps gives its higher coherences. Thus worker names and permutations of equivalent exact reduction plans can be tracked without choosing an order. In the floating-point executable we instead preserve a chosen indexed order before reduction. Floating-point addition has no general associativity law.

6.2 Finite line search and its limits

From the current θ, the coordinator evaluates $\theta - \alpha_k g$ for finitely many positive rates $\alpha_k = \alpha_0/2^k$, with g the assembled gradient. Workers evaluate the candidate losses in parallel. The Lean selector starts with the current measured objective and replaces it only on a strict improvement. A probe with a non-finite scalar output invalidates that trial. An unavailable worker or malformed job fails the batch. The correctness statement uses the same accepted trial whose score was measured.

Proposition 6.2 (Finite acceptance)**.** *For a transitive reflexive score order and a comparator sound for it,* `choose` *returns the original point or one of the finite trials and its score is no greater than the original score.*

Proof. Induct on the list of trials. An accepted head has score no greater than the current one by comparator soundness. The induction hypothesis bounds the result of the tail relative to that new head, and transitivity gives the bound relative to the original point. An unaccepted head leaves the current point unchanged. The membership proof follows the same induction. These arguments are `choose_nonincrease` and `choose_member`. □

If $g = \nabla L(\theta) \neq 0$ for a differentiable real objective, the derivative at zero of $\alpha \mapsto L(\theta - \alpha g)$ is $-\|g\|^2$. Some sufficiently small positive rate therefore decreases the real objective. A finite list of rates can miss it. A finite number of value and derivative queries cannot ensure a global minimum of an arbitrary smooth objective. Indeed, run such an algorithm on the zero function on a nontrivial Euclidean domain. Choose a point separated from the finitely many queries and from the returned point. Subtract a smooth compactly supported bump around that point. All queried jets and the returned point's value are unchanged, while a strictly lower value exists. The executable's guarantee is acceptance of measured improvements over an explicit finite set of candidates.

The implementation computes one forward directional evaluation per parameter coordinate, per sample. If there are p parameters, n samples, and one evaluation costs T, gradient work is $O(npT)$ before scheduling and transmission. This reference backend prioritizes a short proof path and architecture coverage. Tensor fusion, collective GPU kernels, and a reverse-mode scheduler would need separate implementations and proofs about their representations and arithmetic.

7 Architecture examples and reproducible execution

7.1 Encoding neural architecture families

The suite instantiates defining blocks of major architecture families at small fixed shapes. Each instance builds a source term and a scalar loss from the same grammar and follows the same compile, execute, differentiate, and update pipeline. This makes the examples tests of the general compiler entry point. The type-and-semantics proofs are independent of this finite suite. Architecture labels below identify the mechanisms implemented in the examples, whose data and dimensions are specified in `examples/models.py`.

Dense, convolutional, and residual models. The MLP composes two affine maps with a tanh layer and uses squared loss. The CNN uses a shared spatial kernel on a 3×3 input and average pooling, then a readout. The residual CNN adds a two-convolution residual to its input before pooling. Gradients of a shared kernel receive terms from each occurrence in the spatial computation. This exercises differentiation under reuse, the mechanism underlying backpropagation [10], convolutional weight sharing [11], and residual blocks [12].

Recurrent and graph models. The RNN repeats an affine-tanh transition with the same weights over four inputs [13]. The GRU implements reset and update gates [14]. The LSTM propagates a separate cell state through input, forget, and output gates [15]. The graph example uses two degree-normalized message-passing layers and a pooled graph readout, following the normalized adjacency construction in [16]. The local program forms the repeated block or the fixed graph neighborhood at generation time. The compiler also supports repetition over a list of runtime length, exercised separately in the higher-order structural case.

Attention. An attention head computes

$$a_{ij} = \frac{\exp(\langle q_i, k_j \rangle / \sqrt{d})}{\sum_{j' \in M(i)} \exp(\langle q_i, k_{j'} \rangle / \sqrt{d})}, \qquad h_i = \sum_{j \in M(i)} a_{ij} v_j.$$

The decoder uses a causal mask, a three-token vocabulary, and a next-token cross-entropy loss. The encoder allows all positions to attend and uses a classification readout. The encoder-decoder adds source-to-target cross attention to causal target attention. The vision instance patchifies a 4×4 input, adds a learned class token, and applies encoder self-attention. These cover the principal attention wiring patterns of [17, 18, 19, 20]. The executable uses tiny dense projections, two heads of width one, and explicit arithmetic. It exercises softmax, residual paths, normalization, and derivative flow into query, key, value and output projections.

Selective and delta-rule sequence models. The state-space example uses a negative scalar generator $A = -\exp(a)$ and an input-dependent step Δ_t. Its discrete transition and input coefficient are

$$\overline{A}_t = e^{\Delta_t A}, \qquad \overline{B}_t = (\overline{A}_t - 1) A^{-1} B_t.$$

It scans $h_t = \overline{A}_t h_{t-1} + \overline{B}_t x_t$ and gates the state-dependent readout. The other recurrent example updates a matrix memory using a gated delta rule

$$S_t = \alpha_t S_{t-1} + \beta_t (v_t - \alpha_t S_{t-1} k_t) k_t^{\mathsf{T}}, \qquad y_t = S_t q_t.$$

It combines this recurrence with a causal attention block. These are minimal instances of the selective and hybrid mechanisms in [21, 22]. Their execution tests cover input-dependent recurrences and multiplicative state updates.

Mixture of experts. With experts f_j, logits r_j, and active set A, the example returns

$$\sum_{j \in A} \frac{\exp r_j}{\sum_{k \in A} \exp r_k} f_j(x).$$

One program uses all three experts. Another uses a two-of-three route schedule fixed in the source for each token. In the second program only the selected experts are evaluated per token, and the weights among those experts remain trainable. This tests sparse mixtures within a fixed routing region, as occurs when a learned top-k routing choice is stable [23, 24]. At a score tie, differentiation of the discrete route itself requires a separately chosen semantics. The benchmark keeps the routes fixed.

Generative models. Both diffusion losses use noisy data $x_t = \sqrt{1-t}\, x_0 + \sqrt{t}\, \epsilon$ and regress against the fixed sample noise ϵ [25]. One uses convolution, downsampling, a bottleneck, upsampling, and a skip connection in the spirit of U-Net [26]. The other uses 2×2 patches, time-conditioned normalization and a transformer denoiser in the spirit of DiT [27]. The VAE differentiates $z = \mu + \exp(\frac{1}{2} \log \sigma^2)\epsilon$ and a diagonal Gaussian KL term [28]. Two GAN cases use the same small generator and discriminator with either the generator or discriminator parameters frozen [29]. These are separate scalar optimization problems. The affine coupling flow includes its log-Jacobian in the negative log-likelihood [30]. Flow matching regresses a time-conditioned velocity onto $x - z$ at $(1-t)z + tx$ [31].

Multimodal, speech, and recommendation. The CLIP-style case encodes paired synthetic image and text features, normalizes their embeddings with a stabilizer, learns a temperature, and applies symmetric cross-entropy to the full similarity matrix of two pairs [32]. Each such matrix is assembled inside a single objective row, so splitting individual pairs would change the loss. The Conformer-style case combines two half-step feed-forward blocks, attention, gated depthwise temporal convolution and a readout [33]. The recommendation example uses numerical and categorical features, pairwise embedding interactions, and binary cross-entropy [34].

7.2 Higher-order case with a closed-form check

A separate program builds $f = \lambda a.\lambda b.\, ab + a$ and applies it to a trainable parameter w, obtaining a closure h. It folds over a list of values of type real $+ 1$, applying h to the real payload in the left branch and to 1 in the right branch. The cost is the square of the sum. The three sample lists at

$w = 0.2$ contain real 2 and unit, real -0.5 and real 0.3, and the empty list, respectively. The scalar losses are 1, 0.1296, and 0, and the derivatives are 10, 1.296, and 0. This gives the exact batch targets

$$L(0.2) = \frac{1.1296}{3}, \qquad L'(0.2) = \frac{11.296}{3}.$$

The test evaluates these targets through the same wire format and worker pipeline as the neural cases. Additional checks cover the empty eliminator in an unreachable arm, list constructors, both sum tags, a product returned on the wire, and rejection of ill-typed source and corrupted stack code. Another cost, $(x + 1)^2 - 10^{-3} \log x$, begins at $x = 0.1$. The larger trial steps produce non-finite scores and the sixth, of size $1/32$, is finite and strictly decreases the measured objective.

7.3 Measurement protocol and results

We built the executable with Lean 4.19.0 and Lake, then ran Python's coordinator with three independent TCP workers on loopback on Linux x86_64. Each named case used three objective rows and four accepted updates. The starting rate at each iteration was 0.18, with six trial rates obtained by successive halving. Inputs, losses and directions used IEEE binary64, with wire values represented as exact 64-bit patterns. The suite compared worker results with a separate recursive Python interpreter for the source program. For every trainable coordinate, it computed a central difference of that interpreter's scalar cost with $h_j = 10^{-5}(1 + |\theta_j|)$. The reported error for each case is

$$\max_j \frac{|g_j - g_j^{\mathrm{FD}}|}{1 + |g_j^{\mathrm{FD}}|}.$$

The acceptance threshold was 2×10^{-6}. For each case the one-worker and three-worker value-gradient pairs matched bit for bit with the same reduction order, and the reference interpreter also matched the final loss within the declared tolerance of 3×10^{-12} absolute or relative.

The suite evaluated 935 trainable-coordinate comparisons across all 26 cases and accepted 104 updates. The largest scaled discrepancy was 6.62×10^{-10}, in the contrastive example. The Real NVP row omits the Gaussian additive normalization constant, so the reported reduced negative log-likelihood can be negative. Each objective has its own scale and initial condition. Table 1 reports the individual executions, without treating the losses as a cross-model accuracy ranking.

Table 1: Measured objectives and gradient discrepancies on three TCP workers.

Case	Params.	Initial loss	Final loss	Scaled error
MLP	21	0.1239962	0.0213232	1.25×10^{-11}
Spatial CNN	10	0.2692596	0.2325899	1.84×10^{-11}
Residual CNN	12	0.2904074	0.2322504	2.47×10^{-12}
RNN	13	0.0858071	0.0695565	2.18×10^{-12}
GRU	33	0.0596449	0.0565293	2.21×10^{-11}
LSTM	43	0.0879157	0.0683431	1.00×10^{-11}
Decoder transformer	58	1.0777150	1.0268303	1.42×10^{-11}
Encoder transformer	47	1.2418545	0.4833821	1.16×10^{-11}
Encoder-decoder transformer	112	0.7702176	0.7322609	1.22×10^{-11}
Vision transformer	62	0.8795806	0.2972389	1.42×10^{-11}
Graph convolution	15	0.2987850	0.2895769	6.34×10^{-12}
Selective state-space model	10	0.2437729	0.2435133	2.04×10^{-12}
Delta-attention hybrid	68	0.4346769	0.1398037	2.72×10^{-10}
Soft MoE	48	0.2481291	0.1628168	3.45×10^{-12}
Sparse MoE, fixed routes	48	0.2513659	0.1724777	3.05×10^{-12}
Diffusion U-Net	16	0.2563024	0.2374049	4.16×10^{-12}
Diffusion transformer	71	0.2982492	0.1961924	1.95×10^{-10}
VAE	37	0.9860251	0.3802710	1.82×10^{-11}
GAN, generator step	6	0.4330408	0.4329290	2.64×10^{-12}
GAN, discriminator step	13	0.7385610	0.7151317	6.14×10^{-12}
Real NVP	4	0.3146167	-0.0115711	3.37×10^{-11}
Flow matching	35	0.8040606	0.6987641	1.14×10^{-11}
CLIP-style contrastive	25	1.2949897	0.4550934	6.62×10^{-10}
Conformer-style speech block	85	0.3459806	0.0682097	5.42×10^{-12}
DLRM-style recommendation	42	0.6663279	0.6584869	5.54×10^{-12}
Higher-order closure and fold	1	0.3765333	1.12×10^{-7}	5.50×10^{-13}

7.4 Proof audit and reproducibility

The source package contains a machine-readable trace with the loss trajectory, parameter count, finite-difference error, worker count, and SHA-256 digests of both the source specification and emitted code for every case. The independent interpreter and tests are part of the same package. Run from the package root:

```
lake build
lake env lean ProofAudit.lean
python3 -m examples.verify
```

The audit prints the axioms used by the preservation, jet, reduction, and acceptance theorems. The checked proofs use no admitted lemmas. The reported dependencies are Lean's standard `propext` and `Quot.sound` where required. The exact polynomial case also executes a closed proof by definitional equality. The artifact includes a journal body and the source for this manuscript. The accompanying README defines the source grammar, wire conventions, and commands for separate worker processes.

The higher categorical results are mathematical proofs in this manuscript. The Lean kernel checks the stack and logical-relations theorems, the polynomial jet model, and the abstract reduction and acceptance laws. The worker transport, the libm primitives, the JSON codec, and the coordinator are covered by executable checks rather than a verification of OS, network, or binary64 semantics. The measured suite used separate processes on one host. The implementation accepts remote TCP endpoints for a trusted deployment, with the same compiler and worker code.

8 Relationship to prior work and scope

The differentiable language here follows the simply typed style of [1], with algebraic data and a distinguished smooth scalar type. Ehrhard and Regnier's differential λ-calculus is a different formal presentation that includes formal sums and a differential application operator [9]. An expression in that syntax would require an elaboration into this typed source, or a different frontend and preservation proof. The source grammar here has total structural recursion on lists. General recursion and effects require a semantic treatment of partiality or effects and a corresponding operational theorem.

The relationship to ordinary categorical gluing is exact at the monomorphism subcategory of Theorem 3.2. The higher gluing keeps spaces of choices, yields a relative descent statement, and admits examples with automorphisms. Its Taylor construction concerns a finite Weil algebra and a chosen differentiable scalar structure. An arbitrary ∞-topos by itself supplies no such structure. The CHAD approach [3] provides relevant correctness results and a route to more efficient higher-order reverse-mode differentiation. The present executable uses a dual-number interpreter over the same typed stack code for both primal and dual evaluation.

The exact worker reduction concerns replicated evaluation and finite summation. It differs in objective and protocol from federated averaging [35]. In the implementation the user specifies shared parameter coordinates, a finite sample list, and the score to minimize. The examples train miniature differentiable architecture instances. They establish that their defining operations flow through the compiler and execute with the reported numerical evidence. Evaluation at production parameter counts, inference on accelerators, and convergence of large stochastic training are separate questions.

Authorship and automation. Prepared for anonymous review. The preparation of the manuscript, Lean development, and execution suite used AI-assisted drafting and implementation. The evidence accompanying the submission consists of the mathematical arguments, kernel-checkable sources, and reproducible execution records identified above.

References

[1] M. Huot, S. Staton, and M. Vákár. Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing. In *FoSSaCS*, 2020. arXiv:2001.02209v3.

[2] M. Huot, S. Staton, and M. Vákár. Higher Order Automatic Differentiation of Higher Order Functions. arXiv:2101.06757, 2021.

[3] M. Vákár and T. Smeding. CHAD: Combinatory Homomorphic Automatic Differentiation. arXiv:2103.15776, 2021.

[4] J. Lurie. *Higher Topos Theory*. Princeton University Press, 2009. arXiv:math/0608040.

[5] J. Shah. Recollements and stratification. arXiv:2110.06567, 2021.

[6] K. Bauer, M. Burke, and M. Ching. Tangent ∞-categories and Goodwillie calculus. arXiv:2101.07819, 2021.

[7] J. C. Baez and A. E. Hoffnung. Convenient Categories of Smooth Spaces. arXiv:0807.1704, 2008.

[8] N. Gambino and J. Kock. Polynomial functors and polynomial monads. arXiv:0906.4931, 2009.

[9] T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theoretical Computer Science* 309(1–3), 1–41, 2003. DOI.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature* 323, 533–536, 1986. DOI.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* 86(11), 2278–2324, 1998. Publisher.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2015.

[13] J. L. Elman. Finding Structure in Time. *Cognitive Science* 14(2), 179–211, 1990. Article PDF.

[14] K. Cho et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. arXiv:1406.1078, 2014.

[15] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation* 9(8), 1735–1780, 1997. PubMed.

[16] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907, 2016.

[17] A. Vaswani et al. Attention Is All You Need. arXiv:1706.03762, 2017.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805, 2018.

[19] C. Raffel et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683, 2019.

[20] A. Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929, 2020.

[21] A. Gu and T. Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv:2312.00752, 2023.

[22] S. Yang et al. Gated Delta Networks: Improving Mamba2 with Delta Rule. arXiv:2412.06464, 2024.

[23] N. Shazeer et al. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. arXiv:1701.06538, 2017.

[24] A. Q. Jiang et al. Mixtral of Experts. arXiv:2401.04088, 2024.

[25] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. arXiv:2006.11239, 2020.

[26] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597, 2015.

[27] W. Peebles and S. Xie. Scalable Diffusion Models with Transformers. arXiv:2212.09748, 2022.

[28] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. arXiv:1312.6114, 2013.

[29] I. Goodfellow et al. Generative Adversarial Networks. arXiv:1406.2661, 2014.

[30] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. arXiv:1605.08803, 2016.

[31] Y. Lipman et al. Flow Matching for Generative Modeling. arXiv:2210.02747, 2022.

[32] A. Radford et al. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020, 2021.

[33] A. Gulati et al. Conformer: Convolution-augmented Transformer for Speech Recognition. arXiv:2005.08100, 2020.

[34] M. Naumov et al. Deep Learning Recommendation Model for Personalization and Recommendation Systems. arXiv:1906.00091, 2019.

[35] H. B. McMahan et al. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629, 2016.