Higher Monads and Recursive Improvement

Coherent histories, self-modifying policies,
dependent improvement programs, and neural-network symmetries

Anonymous submission draft

November 3, 2026
Version 1.0.0

Prepared for *Higher Categorical Semantics*
https://higher-categorical-semantics.pages.dev

Abstract

Recursive improvement changes a system together with the procedure that proposes its subsequent changes. We give a higher categorical semantics that separates this phenomenon from repeated application of a fixed optimizer. A higher monad is an associative algebra in the composition monoidal infinity-category of endofunctors, with its complete coherent structure retained. We construct a path monad for directed improvement histories and a dependent tree monad for typed recursive improvement programs. A free-monad adjunction supplies a precise metamonad whose multiplication collapses nested improvement languages. We prove conditional preservation, translation, resource, sequential-extension, and geometric base-change results. These algebraic statements are separated from analytic descent and convergence hypotheses. A detailed neural-network model retains exact behavioral fibers, permutation isotropy, function-preserving widening, and updates to learned optimizers. Counterexamples show why monadic coherence alone implies neither strict progress, convergence, generalization, nor an effective universal improver. The contribution is a unified specification and several application-level consequences, not a new proof of the established foundations of higher monad theory. The paper contains mathematical proofs; it makes no machine-checking or benchmark claim.

Keywords. Higher monad; recursive improvement; infinity-category; polynomial functor; dependent type; neural network; learned optimizer; invariant; geometric semantics.

Status and responsibility. This is an unreviewed manuscript prepared with AI assistance at a human request. No acceptance, independent referee assessment, mechanized proof, experimental performance result, or deployed autonomous system is asserted. The submitting researcher is responsible for the final scholarly review and submission decision.

Contents

1 The question and the mathematical contribution **1**
1.1 What is proved . . . 1
1.2 Relation to established work . . . 2

2 Ambient categories, internal logic, and higher coherence **2**
2.1 Universe and existence conventions . . . 2
2.2 Directed arrows and invertible higher cells . . . 2
2.3 What the adjective higher does not imply . . . 3

3 A category of admissible improvement **3**
3.1 Certificates and decidability . . . 4

4 The path higher monad **4**
4.1 Length, evidence, and nontrivial higher structure . . . 6

5 Recursive programs and the tree higher monad **6**
5.1 Dependent polynomial instructions . . . 7
5.2 Finite syntax and unrestricted recursion . . . 7

6 A metamonad for improvement languages **8**
6.1 Objects that represent code . . . 9

7 Operational self-modification **9**
7.1 The state-effect monad . . . 9
7.2 Finite histories and continuing runs . . . 10

8 Invariant lifting and interpretation **10**
8.1 A common carrier for syntax and certified histories . . . 11
8.2 Certified translations between improvers . . . 11

9 Sequential extension of improvement monads **12**
9.1 A language tower is not a numerical trajectory . . . 13

10 Resource grades and quantitative progress **13**

11 Analytic assumptions for descent and convergence **14**
11.1 Why loss convergence is weaker . . . 15
11.2 Stochastic proposals . . . 15

12 A higher category of neural networks **16**

12.1 Interfaces, circuits, and symmetry groupoids . . . 16
12.2 Permutation isotropy is retained . . . 17
12.3 Exact behavioral fibers and still higher homotopy . . . 17

13 Directed learning and architecture rewrites **18**
13.1 A noninvertible exact widening . . . 18
13.2 Residual insertion and typed interfaces . . . 19
13.3 Learning steps descend through permutations . . . 19
13.4 Why arbitrary reparameterization is different . . . 20

14 Recursive improvement of a neural optimizer **20**
14.1 A typed improvement language for this example . . . 21
14.2 Beyond feedforward networks . . . 22

15 Geometric semantics and change of context **22**
15.1 Predicates, descent, and their limits . . . 23

16 Obstructions and counterexamples **23**
16.1 Coherence does not force progress . . . 23
16.2 Self-modification can keep a poor policy . . . 23
16.3 No universal effective strict improver . . . 24
16.4 An objective contract is not a generalization theorem . . . 24
16.5 Changing the checker or the objective . . . 25

17 The combined recursive-improvement specification **25**
17.1 Four different limits . . . 26
17.2 Scope of the proposed theory . . . 26

A A worked nested-program interpretation **26**

B Explicit obligations for a future mechanization **27**

References **28**

1 The question and the mathematical contribution

A system may change its parameters while its update procedure stays fixed. It may instead change the architecture on which that procedure operates, the language in which candidate updates are expressed, or the procedure that proposes the next update. Only the last three possibilities begin to explain the additional structure usually intended by *recursive improvement*. They do not by themselves establish that the changes are improvements. That word requires an objective, an ordering of outcomes, and assumptions relating accepted changes to the ordering.

We therefore study a system with four separately specified components:

state and symmetry, directed changes, recursive syntax, acceptance semantics.

Higher category theory describes symmetries and coherent ways of composing changes. Higher monads describe the substitution and interpretation of programs that contain further improvement programs. A protected specification determines which changes are admissible. Analysis supplies any additional claim about descent or convergence. None of these components can silently replace another.

Our main constructions have two different carriers. The path monad acts on graphs of state changes with fixed state object. The tree monad acts on families of typed program outputs. Both are higher monads, but their units and their algebras have different meanings. Inserting one edge into a free path is not the same operation as returning an unchanged runtime state. Keeping the carriers explicit avoids an ambiguity common in informal accounts of self-improvement.

1.1 What is proved

We establish the following results, with the hypotheses stated at their use. First, finite directed histories form a free associative object under convolution, and this gives a higher monad on graphs. Second, sequentially continuous endofunctors on a presentable infinity-category generate free recursive-program monads by an initial-chain construction. Third, a coherent restriction of an effect monad to an admissible full subcategory preserves admissibility under arbitrary finite Kleisli composition. Fourth, compatible sequential extensions of sequentially continuous monads have a pointwise colimit monad. Fifth, path and finite-arity tree constructions commute with geometric inverse image under the stated base-change hypotheses.

These structural results are supplemented by explicit numerical statements. A sufficient-decrease inequality telescopes along any accepted history. Uniformly controlled gradient steps yield a stationarity bound even when their step-size proposer changes over time. Conditional expected decrease gives a corresponding stochastic bound. Each numerical conclusion has a narrower scope than the structural semantics. A coherent history need not be an improving history, and a convergent loss sequence need not be a convergent sequence of systems.

The neural-network example has a higher space of exact symmetries, nontrivial isotropy, directed architecture changes, and policies that update the optimizer itself. We show precisely which parts of an ordinary gradient update descend through permutation symmetries, and why general coordinate changes require additional metric data.

1.2 Relation to established work

The foundational notion of a monad used here is the associative-algebra definition in Higher Algebra, Section 4.7 [1]. Homotopy coherent adjunctions and monadicity are treated by Riehl and Verity [3]; transformations in the higher categorical setting are developed by Haugseng [4]. Polynomial signatures and free recursive syntax have established categorical foundations [5, 6]. Our contribution is their organization around constrained self-modifying improvement, together with explicit distinctions and examples.

Categorical treatments of learning describe compositional forward and backward information flow [8, 9]. Learned optimizers provide a concrete reason to put optimizer parameters into the evolving state [10]. Function-preserving network expansions provide concrete architecture changes [11]. We use these ideas as examples, without importing their empirical claims into our theorems. We do not claim priority for free monads, higher monads, initial algebras, or categorical neural-network semantics.

2 Ambient categories, internal logic, and higher coherence

2.1 Universe and existence conventions

Fix nested universes sufficient to form the functor categories below. A category without another qualifier is an infinity-category. We write $\mathcal{S}$ for spaces and $\mathrm{Map}_{\mathcal{C}}(x, y)$ for a mapping space. Equivalence is denoted by $\simeq$; it is not an equality of chosen implementations. A presentable category has all small colimits. When a construction uses a sequential colimit, the relevant endofunctor is explicitly required to preserve it.

For internal constructions let $\mathcal{E}$ be an infinity-topos. Slices $\mathcal{E}_{/K}$ are again presentable and locally cartesian closed. Pullback preserves colimits, and filtered colimits commute with finite limits. We use dependent sums and products in these slices. These are structural assumptions about a mathematical universe, not a claim that every internal object has a computable presentation. Background on infinity-topoi, descent, and geometric morphisms is available in [2].

The natural-number object of $\mathcal{E}$ supports ordinary finite recursion. Externally it is the coproduct of countably many terminal objects. Nothing below identifies a natural-number-indexed iteration with an actual completed infinite computation. Higher inductive or homotopical data concern the identifications between terms, not an extra supply of physical execution time.

2.2 Directed arrows and invertible higher cells

An object $S \in \mathcal{E}$ is a higher groupoid of states and state equivalences. Its paths are invertible. A training step that loses information or changes behavior is generally not such a path. Directed evolution instead belongs to an internal category, modeled by a Segal object $\mathcal{R}_\bullet$ with

$$\mathcal{R}_n \simeq \underbrace{\mathcal{R}_1 \times_{\mathcal{R}_0} \cdots \times_{\mathcal{R}_0} \mathcal{R}_1}_{n \text{ factors}}.$$

A complete Segal presentation additionally identifies the equivalences correctly. We use a free Segal presentation when object coordinates must stay fixed and its completion when an infinity-category

rather than that presentation is wanted. Completion is a distinct operation [7]. We do not assume that every fixed-object free-category construction is already complete.

The distinction between levels is essential. A state equivalence can rename neurons. A directed arrow can train weights. A homotopy between directed arrows can identify two presentations of the same certified change. In an $(\infty, 1)$-category, the last comparison is invertible. A noninvertible comparison between algorithms would instead require an $(\infty, 2)$-category or another directed enrichment. We do not call such a comparison an invertible homotopy merely to fit it into the smaller framework.

Definition 2.1 (Higher monad)**.** For a category $\mathcal{C}$, equip $\mathrm{End}(\mathcal{C}) = \mathrm{Fun}(\mathcal{C}, \mathcal{C})$ with its composition monoidal structure. A *higher monad* on $\mathcal{C}$ is an associative algebra object

$$T \in \mathrm{Alg}_{E_1}(\mathrm{End}(\mathcal{C})).$$

Its underlying endofunctor has unit $\eta : \mathrm{id} \Rightarrow T$ and multiplication $\mu : T^2 \Rightarrow T$, together with the full homotopy coherent associative and unital structure.

The displayed formulas

$$\mu \circ T\mu \simeq \mu \circ \mu T, \qquad \mu \circ T\eta \simeq \mathrm{id} \simeq \mu \circ \eta T$$

are only the low-dimensional shadow of that definition. Arbitrary unrelated choices of these homotopies would not suffice. In our constructions, coherence comes from a universal property or from an associative convolution construction, not from checking these three displays alone.

2.3 What the adjective higher does not imply

A higher monad is not automatically an E_2-algebra. The composition tensor is generally nonsymmetric, and updates generally fail to commute. For example, the affine updates $a(x) = 2x$ and $b(x) = x + 1$ satisfy $ab(x) = 2x + 2$ and $ba(x) = 2x + 1$. A braiding that exchanges such updates would require extra mathematical information. The availability of arbitrarily high coherent cells does not provide that information.

Nor does a higher monad automatically describe a hierarchy of increasingly capable agents. It describes composition with coherent substitution. A hierarchy of languages, a hierarchy of optimizers, and the homotopical dimension of their equivalences are three different structures. We will supply precise versions of the first two without identifying them with the third.

3 A category of admissible improvement

Fix an object S of system states. Its coordinates can include an architecture, parameters, optimizer memory, optimizer code, and an audit state. Fix also a specification context $k : K$. It records the objective, admissibility conditions, resource conventions, and observational equivalences. A change of k is not silently counted as improvement relative to the old k.

Let $I : S \to \mathsf{Prop}$ be an admissibility predicate. Here Prop denotes the classifier of propositions, so the fiber $I(s)$ is either empty or contractible in the internal sense. Let $J : S \to Q$ be an objective in an ordered object; externally the main example is $Q = \mathbb{R}$ with lower values preferred. Both I and J must respect the state equivalences represented in S.

Definition 3.1 (Certified primitive change)**.** A graph of certified changes over S is an object

$$G \longrightarrow S \times S$$

whose fiber $G(s,t)$ consists of a primitive change, its declared evidence, and endpoint data. Its evidence includes maps

$$I(s) \longrightarrow I(t), \qquad J(t) \leq J(s),$$

when these two claims are part of the chosen specification. Evidence types may retain additional homotopy; the objective inequality itself is propositional.

We impose neither that the graph contains every mathematically improving pair nor that it contains a nonidentity change at every state. Evidence can be unavailable even when an improvement exists. A deterministic policy is a coherent choice of outgoing certified change, including a permitted idle choice. A stochastic policy additionally needs a declared probability semantics. The graph alone does not choose either policy.

For numerical objectives, let $Q_{\downarrow}$ be the poset category with an arrow $a \to b$ exactly when $b \leq a$. A functor $\mathcal{R} \to N(Q_{\downarrow})$ records monotonicity of all directed arrows. A functor to a poset necessarily sends equivalent objects to equal objective values. Thus an alleged equivalence between systems with different losses cannot coexist with this objective functor.

Example 3.2 (The objective cannot improve by being renamed)**.** Suppose the state contains an editable display field v, and a proposer replaces v by $v - 1$ while leaving the actual network fixed. If J reads v, this is formally improving relative to that definition. If J is the fixed empirical loss of the network, it is an idle behavioral change. The category does not decide which quantity the researcher intended. The specification must define it before the improvement theorem is stated.

3.1 Certificates and decidability

An internal proposition need not have a decision procedure. Therefore an acceptance mechanism cannot always be written as an unexplained instruction to test $I(t)$ and branch on the answer. A constructive alternative is a proposal type

$$\mathsf{Proposal}(s) = 1 + \sum_{t:S} \mathsf{Cert}(s,t).$$

The left summand requests an idle step; the right summand supplies a candidate with the required evidence. A runner only eliminates the supplied sum. It does not decide all assertions about arbitrary programs.

The mathematics proves the behavior of an interpreter that consumes such evidence. An executable checker would require a concrete proof language and a trusted implementation, neither of which is asserted here. Allowing a learned proposer to emit a string labeled as a proof is not an instance of this construction until the evidence interpretation is specified.

4 The path higher monad

Set $\mathsf{Graph}_S(\mathcal{E}) = \mathcal{E}_{/S\times S}$. For graphs G, H, define convolution by

$$(G \otimes_S H)(s,t) = \sum_{u:S} G(s,u) \times H(u,t).$$

This notation denotes a homotopy dependent sum; it is not an ordinary set quotient when S has automorphisms. The unit graph U_S is the diagonal $S \to S \times S$. Associativity follows from the coherent associativity of pullback and dependent sum.

Convolution preserves colimits in each variable: fiber products and dependent sums preserve colimits in the indicated variable in an infinity-topos. Consequently the free associative-algebra construction exists and has underlying graph

$$\mathsf{Path}_S(G) = \coprod_{n \geq 0} G^{\otimes_S n}, \qquad G^{\otimes_S 0} = U_S. \tag{1}$$

Proposition 4.1 (Free coherent histories)**.** *The free-forgetful adjunction between associative convolution algebras and graphs gives a higher monad* Path_S *on* $\mathsf{Graph}_S(\mathcal{E})$*. Its unit inserts an edge as a path of length one. Its multiplication concatenates a finite path of finite paths. Its algebras are associative unital composition structures on graphs over* S*, including all coherent data.*

Proof. For an associative convolution algebra A, a graph map $G \to A$ extends on the nth summand by the n-fold multiplication of A. The associative operad makes the space of compatible extensions contractible. Restriction to the length-one summand is inverse to this extension, yielding the free-forgetful adjunction at the level of mapping spaces. The induced monad therefore carries its full coherent structure.

Concretely, a nested path is indexed by a finite ordered list of finite ordered lists. Flattening forgets the separators and retains the edge order. Repeated flattening has a common ordered-list result. The free associative construction organizes all such flattenings simultaneously, including units and higher associations. Thus the argument is not a choice of one associator followed by an unsupported coherence claim. □

An algebra here presents an internal Segal category with specified object object S. It need not be complete. Its Segal completion gives the associated internal infinity-category. We retain the distinction because completion may change the chosen object presentation, while Formula (1) explicitly fixes it.

Proposition 4.2 (Composition of certificates)**.** *If every primitive edge of* G *preserves* I *and is nonincreasing for* J*, every finite path of certified edges preserves* I *and is nonincreasing for* J*. These assignments are compatible with path flattening and all its coherent associations.*

Proof. For a length-zero path use the identity of $I(s)$ and reflexivity of the order. For a path $(e_1, \ldots, e_n)$ compose the maps of propositions $I(s_{i-1}) \to I(s_i)$ and the inequalities $J(s_i) \leq J(s_{i-1})$. Function composition and order transitivity supply an associative interpretation. Since the order target is a poset and the predicate target is propositional, the compatibility data in these targets are unique when they exist. Interpreting the additional, nonpropositional evidence still uses its specified convolution-algebra structure. The free property extends the primitive interpretation and gives compatibility with multiplication. □

This proposition is conditional. It does not produce a primitive certificate, prove completeness of a checker, or ensure that a selected path has positive length. Its practical content is modular: local obligations are sufficient to compose a large finite history without restating its entire global argument.

4.1 Length, evidence, and nontrivial higher structure

The length map $\mathsf{Path}_S(G) \to \mathbb{N}$ sends concatenation to addition. Evidence spaces need not be discrete. Two primitive rewrites can be related by a path of exact semantic rewrites; that path induces a path between the corresponding histories. Automorphisms of states also contribute to the homotopy dependent sums at intermediate endpoints. Replacing every such space by its set of components discards this information.

Length-zero paths are identity arrows in the generated category. The monad unit itself has length one because its input is already an edge. This distinction matters when the same manuscript also uses a state-effect monad whose unit returns an unchanged state. We reserve η^{Path} for edge insertion and η^T for an effect return.

5 Recursive programs and the tree higher monad

Let $\mathcal{C}$ be presentable and let $P : \mathcal{C} \to \mathcal{C}$ preserve sequential colimits. Think of P as a layer of typed improvement instructions with slots for subprograms. For each output object X, define

$$F_X(Y) = X \amalg P(Y), \qquad Y_0 = 0, \qquad Y_{n+1} = F_X(Y_n).$$

The arrows in the chain are induced by the unique map $0 \to F_X 0$. Put

$$T_P X = \operatorname*{colim}_{n<\omega} Y_n. \tag{2}$$

The X summand is a returned output. A P node is an instruction whose arguments are smaller programs. Branches are typed by the signature, rather than being arbitrary untyped calls back into an evaluator.

Lemma 5.1 (Initial-chain construction)**.** *There is a canonical equivalence $X \amalg P(T_P X) \simeq T_P X$. With this structure, $T_P X$ is initial among F_X-algebras.*

Proof. Since F_X preserves sequential colimits, it sends the chain colimit to the colimit of the shifted chain. The shift is cofinal, giving the displayed equivalence. For an algebra $a : X \amalg P(A) \to A$, the initial stage maps uniquely to A. Inductively, a compatible map $Y_n \to A$ extends to Y_{n+1} by applying P and then a. These maps define a cone and hence a map $T_P X \to A$.

The construction applies to compatible families and all their homotopies: the space of compatible algebra cones is contractible, starting from the contractible choice at 0. The sequential universal property identifies algebra maps out of the colimit with these compatible cones. Thus the result is homotopy initiality, not merely uniqueness of a map in the homotopy category. □

Theorem 5.2 (Free recursive-program monad)**.** *The assignment $X \mapsto T_P X$ is the free higher monad on P. Its unit inserts a returned output. Its multiplication substitutes a program for each returned program. It coherently evaluates a finite nesting of improvement programs without assigning mathematical significance to chosen parentheses.*

Proof. The initial-algebra property makes the construction functorial in X. Given a monad T and a natural transformation $P \Rightarrow T$, interpret a returned output by η^T and a P node by that transformation followed by the multiplication of T. Initiality extends the interpretation to a natural

transformation $T_P \Rightarrow T$. It is compatible with substitution because the two candidate interpretations of a substituted program agree on returns and on one instruction layer; the initial property identifies them coherently.

Conversely, a monad morphism from T_P restricts along the one-layer inclusion $P \Rightarrow T_P$. Extension and restriction give inverse equivalences of mapping spaces. This is the free-monad universal property. Applying it to nested syntax gives substitution multiplication, and the adjunction supplies all unit and associativity coherences. This construction belongs to the established theory of free higher monads; the interpretation in terms of improvement programs is the present specialization. □

5.1 Dependent polynomial instructions

For a concrete internal signature in $\mathcal{E}$, choose a diagram

$$K \xleftarrow{s} E \xrightarrow{p} B \xrightarrow{t} K$$

and define $P = \Sigma_t \Pi_p \Delta_s$ on $\mathcal{E}_{/K}$. The fiber B_k specifies instructions returning color k, and E_b specifies the input slots of instruction b, with input colors determined by s. In element notation,

$$(PX)_k = \sum_{b:B_k} \prod_{e:E_b} X_{s(e)}.$$

This expression is interpreted internally, retaining the symmetries of the signature.

Assume for the results that follow that p is finite locally constant with discrete fibers. Locally the dependent product is a finite product, so it preserves filtered, hence sequential, colimits. Pullback and dependent sum preserve all colimits. Thus P satisfies the initial-chain hypothesis. This assumption excludes unrestricted infinitary branching; no assertion about arbitrary dependent products is hidden in the construction.

The colors can include an architecture type, the admissibility context, and whether an output is a candidate network, an optimizer, or a program that produces optimizers. Instructions can include sequential composition, a finite candidate comparison, and execution of a supplied certified optimizer update. A dependent type can insist that the output optimizer accepts the exact architecture emitted by its preceding subprogram.

5.2 Finite syntax and unrestricted recursion

The terms of $T_P X$ are generated at finite stages. Under the finite-arity hypothesis they are finite well-founded instruction trees. They can describe one step of a continuing controller, but they do not contain a completed infinite execution. General recursion requires additional semantics for partiality, domain-theoretic limits, guarded recursion, or coalgebraic behavior. It is not a consequence of free-monad notation.

An interpreter $T_P X \to TX$ must supply a lawful interpretation for every primitive instruction. If a constructor invokes a learned proposer, the interpreter must still specify how its output is accepted. If a constructor edits optimizer code, it must state which typing and acceptance obligations are preserved. The type structure makes those questions expressible; it does not answer them without evidence.

6 A metamonad for improvement languages

Let $\mathrm{End}_\omega(\mathcal{C})$ be the category of endofunctors preserving sequential colimits, and let $\mathrm{Mnd}_\omega(\mathcal{C})$ be the category of monads with such underlying endofunctors. The construction in Section 5 supplies an adjunction

$$F : \mathrm{End}_\omega(\mathcal{C}) \;\rightleftarrows\; \mathrm{Mnd}_\omega(\mathcal{C}) : U, \qquad F(P) = T_P.$$

Here the morphisms on the right are coherent monad morphisms. To see that the adjunction stays within these categories, observe inductively that each finite stage $Y_n(X)$ preserves sequential colimits in X. Sequential colimits can be interchanged with each other, so the colimit T_P does as well.

Definition 6.1 (Free-language metamonad)**.** The adjunction induces the higher monad

$$\mathbb{M} = UF : \mathrm{End}_\omega(\mathcal{C}) \to \mathrm{End}_\omega(\mathcal{C}).$$

Its unit is the one-instruction inclusion $P \Rightarrow T_P$. Its multiplication is

$$\mathbb{M}^2(P) = T_{T_P} \xrightarrow{U\epsilon_{F(P)}} T_P = \mathbb{M}(P).$$

A node of T_{T_P} is labeled by an operation that is itself a P-program. The multiplication substitutes those inner programs into the outer program. Thus there are two kinds of substitution: $T_P^2 X \to T_P X$ substitutes output programs, whereas $T_{T_P} X \to T_P X$ expands program-valued operation labels. Both are lawful, but they live at different categorical levels.

This is an explicit meaning of a monad acting on a language of improvement procedures. It does not rely on an informal instruction to apply a monad to itself. The carrier of $\mathbb{M}$ is the category of endofunctors. We make no claim that this carrier is the same as the state category $\mathcal{C}$, or that $\mathbb{M}$ is an E_2 structure on one of its objects.

Proposition 6.2 (Conservative interpretation of nested languages)**.** *Let $\rho : P \Rightarrow T$ interpret primitive instructions in a higher monad T. There is a canonical interpretation of every finite metamonadic expansion $\mathbb{M}^n(P)$ in T. Expanding nested labels before interpretation and interpreting their labels first give coherently equivalent results.*

Proof. The free-monad property extends ρ to a monad morphism $\bar{\rho} : T_P \Rightarrow T$. Reapply that property to $\bar{\rho}$ as an interpretation of the expanded operation endofunctor. At the next stage, both routes restrict to ρ on the original operations and respect the same substitution. The adjunction identifies their compatible extension space. Iteration gives the claim, while its higher compatibility is exactly the unit-counit coherence of $F \dashv U$. □

Metamonadic expansion supplies more syntactic descriptions of composite operations. It does not establish that the descriptions express a better optimizer, a new computable function, or a larger observable behavior space. Already T_P can express finite P-programs. Expanding such programs as new node labels can be semantically redundant. Strict language enrichment instead requires genuinely new interpreted operations or a changed resource model.

Example 6.3 (Adding a learned proposer)**.** Let P_0 contain a fixed gradient instruction and an idle instruction. Add to obtain P_1 an instruction whose supplied optimizer parameters determine a candidate step, with an unchanged acceptance interface. A transformation $P_0 \Rightarrow P_1$ induces $T_{P_0} \Rightarrow T_{P_1}$. This extension differs from $\mathbb{M}(P_0)$: it adds a primitive source of candidate changes. Whether its accepted traces outperform the old language remains a separate assertion.

6.1 Objects that represent code

Self-modification requires a specified code object O and an evaluation map on a chosen class of well-typed code. It does not require an evaluator for every total computable function. For example, O can encode finite neural-network architectures with bounded primitive instructions and finite parameter arrays. Its interpretation is then defined by finite structural recursion.

If arbitrary looping programs are admitted, the interpretation must account for partiality or supply a separate termination argument. Treating all syntax as denoting total functions would change the theory. The higher categorical presentation preserves this distinction; it does not remove the computability obstruction described later.

7 Operational self-modification

Write a typical state as

$$s = (a, w, o, m, h) \in \sum_{a:\mathsf{Arch}} W_a \times O_a \times M_a \times H_a.$$

The components are architecture, model parameters, optimizer representation, optimizer memory, and retained history. The products can be dependent if an optimizer's memory shape depends on its representation. The specification context k is a parameter of this type, not an editable field whose meaning can be changed without re-establishing the specification.

A fixed optimizer changes w and perhaps m, while o stays constant. A self-modifying step changes o and may change a. A *recursive* self-modifying step lets the current o participate in proposing its successor. The state transition law then has the form

$$\mathsf{propose}(o, s) : \mathsf{Proposal}(s), \qquad u(s) = \mathsf{run}(\mathsf{propose}(o, s)).$$

The protected runner implements the fixed evidence-elimination interface. It may be mathematically simple while the proposer is highly expressive.

Calling u self-modifying does not mean that u is an untyped function whose own definition changes during evaluation. It means that u reads an explicit component o of the state and can write its successor o'. The outer transition semantics stays defined. Higher syntax additionally allows programs that generate programs updating o.

7.1 The state-effect monad

In a cartesian closed setting, the total state monad is

$$\mathsf{St}_S(X) = (X \times S)^S.$$

Its unit sends x to $s \mapsto (x, s)$. If $H : S \to \mathsf{St}_S(X) \times S$, multiplication first evaluates $H(s) = (k, s')$ and then evaluates $k(s')$. A writer monoid L can also be retained:

$$\mathsf{St}_{S,L}(X) = (X \times S \times L)^S.$$

The multiplication multiplies log values in execution order. These formulas specify monads by the usual cartesian closed adjunctions and state composition; their higher coherence is inherited from those constructions.

A total updater $u : S \to S$ is a global element of $\mathsf{St}_S(1)$. It may update its optimizer component. Thus self-modification is compatible with an ordinary state-effect shape; the specifically higher contribution is the coherent structure of states, rewrites, interpretations, and program substitution.

For admissible states $S_I = \sum_{s:S} I(s)$, a runner can be typed directly over S_I. Its state monad is St_{S_I}. This should not be confused with a restriction of St_S to some arbitrary class of output objects. Changing the state object and restricting an endofunctor's domain are different operations.

7.2 Finite histories and continuing runs

Given a total u and s_0, ordinary natural-number recursion defines $s_{n+1} = u(s_n)$ and every finite prefix. A continuing controller can therefore have well-defined finite observations without the sequence converging. A terminal coalgebra or a space of streams, when used, represents the continuing behavior. It does not automatically contain a limiting optimized state.

The existence of an outgoing edge at every state also does not by itself supply a coherent global policy, particularly in an internal logic without a choice principle. Our deterministic results assume a selected transition map or a selected sequence of certified steps. They do not infer that map from a collection of merely inhabited fibers.

8 Invariant lifting and interpretation

We now give a general effect-level preservation statement that is independent of any particular state implementation.

Definition 8.1 (Admissible lift)**.** Let $j : \mathcal{C}_I \hookrightarrow \mathcal{C}$ be fully faithful. An admissible lift of a monad T is a monad T_I on $\mathcal{C}_I$ and a coherent equivalence

$$Tj \simeq jT_I$$

compatible with units, multiplication, and all their higher coherence.

Full faithfulness is useful because compatibility diagrams can be checked after j. If T preserves the essential image of j, its unit and multiplication restrict through j. Their lift is unique up to the relevant contractible choices; the resulting structure is the admissible lift. Merely writing an objectwise isomorphism without coherent naturality is insufficient.

Theorem 8.2 (Finite preservation under recursive interpretation)**.** *Assume an admissible lift T_I, a presentable category $\mathcal{C}_I$, and a signature P preserving $\mathcal{C}_I$ with induced restriction P_I. Assume P_I preserves sequential colimits. Suppose every primitive instruction is interpreted by a natural transformation $P_I \Rightarrow T_I$. Then:*

1. *every finite P_I-program has a canonical interpretation in T_I;*
2. *substitution and all finite metamonadic expansions preserve that interpretation;*
3. *every finite Kleisli composite of interpreted programs remains in $\mathcal{C}_I$.*

Proof. The free property extends the primitive transformation to a monad morphism $T_{P_I} \Rightarrow T_I$. A monad morphism respects unit and substitution with their coherent structure. Section 6 then supplies compatibility with finite metamonadic expansion.

For the final statement, two Kleisli arrows $f : X \to T_I Y$ and $g : Y \to T_I Z$ compose as

$$X \xrightarrow{f} T_I Y \xrightarrow{T_I g} T_I^2 Z \xrightarrow{\mu_I} T_I Z.$$

Every object and arrow in this display belongs to $\mathcal{C}_I$. Induction covers any finite composite. Applying j identifies the resulting execution with its interpretation in $\mathcal{C}$. Associativity of the monad identifies all coherent bracketings of the same composite. □

This theorem is a closure theorem. The evidence for primitive instructions and the admissible lift are hypotheses, not conclusions. In the state model, one can instantiate the theorem using typed admissible states or use the direct path-certificate argument of Section 4. A particular application should state which route it uses.

8.1 A common carrier for syntax and certified histories

There is an explicit bridge between the path and tree monads that avoids confusing their carriers. Fix a certified graph G over S. On $\mathcal{C} = \mathsf{Graph}_S(\mathcal{E})$ define

$$\mathsf{H}_G(X) = \mathsf{Path}_S(G \amalg X).$$

This is the monad for freely adjoining composable variable edges X to the fixed primitive edges G. Its adjunction has, on the algebra side, internal composition structures equipped with a chosen graph map from G. The free object on X is the category generated by $G \amalg X$.

The unit inserts a variable edge. Multiplication in $\mathsf{Path}_S(G \amalg \mathsf{Path}_S(G \amalg X))$ interprets an outer G edge as its one-edge history and an outer variable as its supplied inner history, then concatenates. The adjunction supplies coherence. A closed interpretation has no unresolved variable edges:

$$\mathsf{H}_G(0) = \mathsf{Path}_S(G).$$

Take the colors of a dependent program signature to be pairs (s, t), so that a program denotes a history from s to t. A certified primitive is a nullary constructor of its endpoint color. A composition constructor with intermediate state u has two input colors $(s, u), (u, t)$ and output color (s, t). These data define a finite-arity polynomial P on the same graph category. Interpreting its primitive labels and composition constructors gives a natural transformation $P \Rightarrow \mathsf{H}_G$, hence a monad morphism

$$T_P \longrightarrow \mathsf{H}_G.$$

At 0 it interprets every closed typed program as a certified finite history. Runtime evaluation takes the target endpoint of that history. The tree-monad return denotes an unresolved graph variable when variables are present; it is not being identified with a state vertex. This bridge is the history interpretation used in the combined theorem and the neural example.

8.2 Certified translations between improvers

For monads T, T' on the same category, a coherent monad morphism $\alpha : T \Rightarrow T'$ translates a T-Kleisli arrow $f : X \to TY$ to $\alpha_Y f : X \to T'Y$.

Proposition 8.3 (Translation respects execution)**.** *The translation above defines a functor between Kleisli categories that is the identity on objects. If admissible lifts are supplied and α restricts between them, the translation preserves admissibility.*

Proof. The unit condition gives identities. For composition the monad-morphism condition, naturality of α, and the multiplication comparison give

$$\alpha_Z \, \mu^T_Z \, T(g) \, f \simeq \mu^{T'}_Z \, T'(\alpha_Z g) \, \alpha_Y f.$$

These are the two translated composites. The coherent morphism, rather than only the displayed comparison, provides compatibility for longer strings. Restricting this reasoning to $\mathcal{C}_I$ proves the second assertion. □

Translation does not automatically preserve loss or cost. To obtain those conclusions it must also preserve their interpretations. A compiler that preserves only the returned function can change runtime cost; an architecture rewrite that preserves outputs can change the optimizer's future dynamics.

9 Sequential extension of improvement monads

Suppose a sequence of higher monads and coherent monad morphisms is given:

$$T_0 \longrightarrow T_1 \longrightarrow T_2 \longrightarrow \cdots .$$

Assume every T_n preserves sequential colimits. Define $T_\infty X = \operatorname{colim}_n T_n X$, pointwise in X. This is a candidate for one language incorporating the compatible operations at all finite stages.

Theorem 9.1 (Sequential-extension monad)**.** *The pointwise endofunctor T_∞ preserves sequential colimits and has a canonical higher monad structure for which the maps $T_n \to T_\infty$ form a colimit cocone of monads in this class.*

Proof. Interchanging two sequential colimits proves preservation. For multiplication, there are canonical equivalences

$$\begin{aligned} T_\infty T_\infty X &\simeq \operatorname*{colim}_i T_i\big(\operatorname*{colim}_j T_j X\big) \\ &\simeq \operatorname*{colim}_{i,j} T_i T_j X \\ &\simeq \operatorname*{colim}_k T_k T_k X \longrightarrow \operatorname*{colim}_k T_k X. \end{aligned}$$

The middle equivalence uses preservation by each T_i. The diagonal $\mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is cofinal: beyond any pair of indices the diagonal tail is nonempty and directed. The last arrow is induced by the compatible multiplications. Units give the unit of the colimit.

For any finite number of factors the same diagonal argument applies. The units, all associative operations, and their coherent relations therefore pass along the cocone of associative algebra objects. Equivalently, the composition tensor in $\mathrm{End}_\omega(\mathcal{C})$ preserves sequential colimits in each variable, so this sequential sifted colimit of associative algebras is created on underlying objects. This supplies the full coherence, beyond the binary formula.

Finally, mapping a pointwise colimit cocone to a target endofunctor is equivalent to giving compatible maps from its stages. Compatibility with each finite associative operation can be tested on the same cofinal diagonal. Thus a compatible cocone of monad morphisms extends uniquely as a coherent monad morphism. This proves the stated universal property. □

Corollary 9.2 (Admissibility through extension)**.** *If each T_n preserves a full subcategory $\mathcal{C}_I$ and its inclusion creates sequential colimits, then T_∞ preserves $\mathcal{C}_I$. The compatible restricted monads supply the admissible lift of T_∞.*

Proof. For $X \in \mathcal{C}_I$, the sequence $T_n X$ lies in $\mathcal{C}_I$. Its colimit computed there is sent by the inclusion to the pointwise colimit in $\mathcal{C}$. Restrict the unit and every multiplication along the resulting fully faithful inclusion. □

The hypothesis on colimits cannot be omitted. A class of bounded-size models, for example, is generally not closed under a growing union of architectures. Nor does this theorem assert that J of a categorical colimit is the limit of the values $J(T_n X)$. That would require continuity assumptions on a separately specified objective.

9.1 A language tower is not a numerical trajectory

An object T_∞ combines operations. A run $s_0 \to s_1 \to \cdots$ records selected executions. Neither determines the other without an interpretation and a policy. Adding a useful primitive operation can improve the available language while a poor policy never invokes it. A fixed language can also support a long trajectory of strict numerical improvements. The two phenomena should not share an unqualified convergence claim.

10 Resource grades and quantitative progress

Let $c : G \to [0, \infty]$ assign a nonnegative resource cost to primitive edges, invariant under their equivalences. Interpret a history's cost as the sum of its edge costs. This sum includes any explicitly modeled cost of proposal generation, evidence checking, and execution; omitting a cost from the model does not bound it in an implementation.

For $r \geq 0$, let $\mathsf{Path}_{\leq r}(G)$ be the subobject of paths with cost at most r. Concatenation restricts to

$$\mathsf{Path}_{\leq r}(G) \otimes_S \mathsf{Path}_{\leq s}(G) \longrightarrow \mathsf{Path}_{\leq r+s}(G).$$

The length-zero unit has cost zero. These restrictions give a graded composition structure. Unless $r = 0$ or a different closure property is supplied, $\mathsf{Path}_{\leq r}$ is not itself an ordinary monad closed under its own multiplication. Executing two budget-r programs can spend $2r$.

Proposition 10.1 (Telescoping certificates)**.** *Suppose an accepted finite trajectory satisfies*

$$J(s_{n+1}) \leq J(s_n) - a_n + b_n, \qquad a_n, b_n \geq 0,$$

and $J \geq J_*$ on its states. Then for every N,

$$\sum_{n=0}^{N-1} a_n \leq J(s_0) - J_* + \sum_{n=0}^{N-1} b_n.$$

If the trajectory is infinite and $\sum_n b_n < \infty$, then $\sum_n a_n < \infty$ and $a_n \to 0$.

Proof. Sum the inequalities. Intermediate values cancel, leaving $J(s_N)$ on the left. Use its lower bound. In the infinite case, the partial sums of the nonnegative a_n are bounded, so their series converges and its terms tend to zero. □

The terms b_n permit a declared approximation allowance. With $b_n = 0$ the loss is nonincreasing. With nonzero b_n, the theorem only bounds cumulative progress loss; individual steps may increase J. A claimed monotonicity theorem must not cite this version without setting $b_n = 0$.

Multiple resources and objectives can be kept as vectors. Componentwise inequalities compose by addition and transitivity. A weighted scalar objective is also possible but changes the preference relation. In particular, a function-preserving network widening has zero loss change and positive parameter cost; it is not a strict Pareto improvement under both criteria.

11 Analytic assumptions for descent and convergence

The structural theory places no topology or differentiable structure on a numerical optimizer unless one is declared. For this section let $L : \mathbb{R}^d \to \mathbb{R}$ be continuously differentiable. Suppose its gradient is M-Lipschitz on every line segment used by the updates. This local segment condition suffices for the descent inequality.

Lemma 11.1 (Segment descent)**.** *For a step $w' = w - \eta \nabla L(w)$ with $\eta > 0$,*

$$L(w') \le L(w) - \eta \left(1 - \frac{M\eta}{2}\right) \|\nabla L(w)\|^2 .$$

Proof. Set $v = w' - w$. The fundamental theorem of calculus gives

$$L(w + v) - L(w) = \int_0^1 \langle \nabla L(w + tv), v \rangle \, dt.$$

Subtract $\langle \nabla L(w), v \rangle$ and bound the remaining integrand in absolute value by $Mt \|v\|^2$. Integration yields the quadratic upper bound $L(w + v) \le L(w) + \langle \nabla L(w), v \rangle + (M/2) \|v\|^2$. Substitute $v = -\eta \nabla L(w)$. □

Theorem 11.2 (Changing proposers with a common descent contract)**.** *Suppose every executed step is $w_{n+1} = w_n - \eta_n \nabla L(w_n)$, with*

$$0 < \underline{\eta} \le \eta_n \le \overline{\eta} < 2/M, \qquad L(w_n) \ge L_*.$$

The rule choosing η_n may change its own parameters and memory arbitrarily. Define $c = \underline{\eta}(1 - M\overline{\eta}/2) > 0$. Then

$$\sum_{n=0}^{N-1} \|\nabla L(w_n)\|^2 \le \frac{L(w_0) - L_*}{c}, \qquad \min_{n<N} \|\nabla L(w_n)\|^2 \le \frac{L(w_0) - L_*}{cN}.$$

For an infinite run, $\|\nabla L(w_n)\| \to 0$. Every cluster point is stationary.

Proof. Segment descent bounds the coefficient of the gradient norm below by c. Apply Proposition 10.1 with $a_n = c \|\nabla L(w_n)\|^2$ and $b_n = 0$. The minimum bound follows by comparing the sum to N times its least summand. Convergence of the nonnegative series implies that its terms vanish. If a subsequence converges to w_*, continuity of ∇L gives $\nabla L(w_*) = 0$. □

The theorem constrains every executed step, not merely the proposals that happened to be accepted during a finite experiment. If a controller is permitted to idle forever away from stationarity, the conclusion need not hold. For such a controller the same theorem applies to its accepted-update subsequence only if that subsequence is infinite and satisfies the stated bounds. We do not infer a lower bound on acceptance frequency.

The theorem also fixes the parameter space and the loss during its run. Changing architecture requires a common analytic state space or a separate argument between architecture changes. Function-preserving expansion transfers the current loss value, but it does not by itself transfer the Lipschitz constant or prove stationarity in the enlarged space.

11.1 Why loss convergence is weaker

A bounded decreasing sequence of real loss values converges. Its parameter sequence can fail to converge. For a simple compact example, take states $(r, z) \in [0, 1] \times S^1$, objective $J(r, z) = r$, and

$$r_n = \frac{1}{n+1}, \qquad z_n = \exp(i \log(n+1)).$$

The loss strictly decreases to zero. For any fixed angle θ, choosing integers close to $\exp(2\pi k + \theta)$ gives a subsequence with $z_n \to e^{i\theta}$. Hence the state sequence has many cluster points. This example is a counterexample to loss-implies-state convergence, not to Theorem 11.2.

A genuine fixed-point conclusion can be obtained from a different hypothesis. If a deterministic update u on a complete metric state space is a contraction with factor $q < 1$, then

$$d(s_n, s_{n+m}) \le \frac{q^n}{1-q} d(s_0, s_1).$$

The sequence is Cauchy, its limit s_* satisfies $u(s_*) = s_*$ by continuity, and two fixed points must have distance at most q times that distance, hence coincide. Self-modification is allowed only if the augmented state update satisfies the contraction assumption. No monad law supplies that inequality.

11.2 Stochastic proposals

For probabilistic runs we work externally on a filtered probability space, or in a separately supplied category of measurable kernels. An arbitrary infinity-topos is not presumed to carry a canonical probability monad. Let J_n, a_n, b_n be nonnegative adapted integrable variables satisfying

$$\mathbb{E}[J_{n+1} \mid \mathcal{F}_n] \le J_n - a_n + b_n.$$

Assume $\mathbb{E}J_0 < \infty$ and $\sum_n \mathbb{E}b_n < \infty$.

Proposition 11.3 (Expected cumulative progress)**.** *For every N,*

$$\sum_{n<N} \mathbb{E}a_n \le \mathbb{E}J_0 + \sum_{n<N} \mathbb{E}b_n.$$

Consequently $\sum_n a_n < \infty$ almost surely. In particular $a_n \to 0$ almost surely.

Proof. Take expectations, use the tower property, and telescope. Nonnegativity permits dropping $\mathbb{E}J_N$. Monotone convergence bounds $\mathbb{E}[\sum_n a_n]$ by the finite right-hand side. A nonnegative extended random variable with finite expectation is finite almost surely. A convergent series of nonnegative terms has terms tending to zero. □

This result survives adaptive changes of the proposer because the inequality is conditional on the entire past. It does not justify the inequality for a particular learned policy. Nor does it prove generalization, a global optimum, or that a randomly accepted step is individually improving. Those claims require additional hypotheses.

12 A higher category of neural networks

We now construct the example that motivates the abstract machinery. We first fix exact symmetries, then introduce directed learning. A continuous curve of weight vectors that changes the represented function is not treated as an exact behavioral equivalence. This restriction is necessary for a loss function to remain meaningful.

12.1 Interfaces, circuits, and symmetry groupoids

An interface is a finite-dimensional real vector space with specified coordinates, written $\mathbb{R}^d$. A neural circuit $A : d \to e$ is a finite typed circuit built from affine maps, a fixed smooth componentwise activation, copying, and addition. Parallel composition and ordinary composition are allowed. A finite circuit has a parameter space $W_A \simeq \mathbb{R}^{p_A}$ and a smooth evaluation

$$F_A : W_A \times \mathbb{R}^d \longrightarrow \mathbb{R}^e.$$

Using smooth activations makes the differentiable statements literal. Piecewise smooth activations require an appropriately qualified version of the analytic section.

Circuit isomorphisms preserve external interfaces and primitive operations. They can relabel hidden coordinates, with the corresponding action on parameters. For a chosen representative A, let G_A be its finite group of such relabelings. Every $g \in G_A$ satisfies

$$F_A(gw, x) = F_A(w, x).$$

We retain the action groupoid, not just its orbit set.

Define a bicategory $\mathcal{B}_{\mathrm{nn}}$ whose objects are interfaces, whose 1-cells are parameterized circuit instances (A, w), and whose invertible 2-cells are these exact circuit symmetries. Horizontal composition glues interfaces and combines parameter tuples. Vertical composition composes relabelings. One may choose strict circuit syntax in a free strict monoidal presentation; alternatively the canonical associators for finite circuit gluing satisfy the bicategory coherence conditions.

Taking the nerve of each hom groupoid gives Kan mapping spaces. A strict presentation followed by the homotopy coherent nerve therefore gives an infinity-category $\mathcal{N}_{\mathrm{sym}}$. Its parallel circuit tensor is symmetric monoidal. This is a genuine higher categorical example: its 2-cells need not be identities. It is a special case with groupoidal, 1-truncated mapping spaces; we do not claim nonzero homotopy in every dimension.

The construction also admits smooth families. Take the infinity-topos of sheaves of spaces on a site of Euclidean test spaces with open-cover descent. Parameter spaces are representable, 0-truncated sheaves. The quotient stack $[W_A/G_A]$ retains permutations and their descent data. Smooth curves are families over an interval; they are not automatically identity paths between unequal parameter values in this topos. This avoids conflating variation of parameters with equality of behavior.

12.2 Permutation isotropy is retained

For a one-hidden-layer network of width m, write

$$F_{A,B,a,b}(x) = B\,\sigma(Ax + a) + b.$$

For an $m \times m$ permutation matrix Π, define

$$A' = \Pi A, \qquad a' = \Pi a, \qquad B' = B\Pi^{-1}, \qquad b' = b.$$

Since $\sigma(\Pi y) = \Pi\sigma(y)$, direct substitution proves $F_{A',B',a',b'} = F_{A,B,a,b}$ on every input.

At the all-zero parameter tuple, every permutation fixes the tuple. Its automorphism group is therefore S_m. The corresponding component of the parameter groupoid has a loop group S_m at that point. Replacing the groupoid by its set of orbits would keep one zero-network point but erase its nontrivial automorphisms. In the neural category, these automorphisms occur as nontrivial higher cells between a network 1-cell and itself.

This matters for descent of an optimizer. A rule defined on labeled coordinates must carry isomorphic parameter tuples to isomorphic outputs. Otherwise the resulting training process depends on a labeling that the state semantics declared irrelevant. The presence of higher cells gives a precise naturality obligation.

12.3 Exact behavioral fibers and still higher homotopy

There is a second, richer enrichment when additional exact homotopies are desired. For a represented function f, let

$$W_{A,f} = \{w \in W_A : F_A(w, -) = f\}.$$

Give this fiber its subspace topology, or its compactly generated replacement. Use mapping spaces

$$\coprod_{[A]:d\to e} \coprod_{f} \mathrm{Sing}(W_{A,f} /\!\!/ G_A). \tag{3}$$

The double slash denotes homotopy quotient; the coproduct over f is discrete. Composition of circuits and of their functions gives compatible maps on these spaces. Keeping canonical graph-gluing coherence gives a homotopy coherent enriched category and hence an associated infinity-category, denoted $\mathcal{N}_{\mathrm{exact}}$.

All homotopies within a summand preserve the complete function f. In contrast, applying the singular functor to the whole unconstrained Euclidean parameter space would identify paths of networks whose losses can differ. Such an indiscriminate enrichment would not support our objective as a map to a discrete ordered set.

Mapping spaces in Formula (3) can have higher topology. For example, enlarge the exact symmetry groups to allow a deep linear two-factor circuit $F_{A,B}(x) = BAx$, with a continuous internal gauge group $G = GL_m^+(\mathbb{R})$ acting as

$$(A, B) \longmapsto (gA, Bg^{-1}).$$

The zero-function fiber contains the fixed point $(0, 0)$. The Borel projection $W_{A,0}//G \to BG$ has the section induced by that fixed point. Thus BG is a homotopy retract of this exact semantic mapping space. Polar decomposition contracts the positive-definite factor of $GL_m^+(\mathbb{R})$ and retracts it to $SO(m)$. For $m = 2$, the universal principal bundle therefore gives $\pi_2(BG) = \pi_1(G) \cong \mathbb{Z}$, which injects into the corresponding homotopy group of the mapping space. Here the higher enrichment retains information beyond an ordinary groupoid.

An optimizer that is natural for $\mathcal{N}_{\text{sym}}$ need not be natural for $\mathcal{N}_{\text{exact}}$. In particular, two parameterizations representing the same current function can train to different future functions. We therefore use $\mathcal{N}_{\text{sym}}$ for the basic training-state quotient, and invoke a richer quotient only when its additional equivariance obligations are met. Exact function equality is sufficient for loss invariance, but insufficient for optimizer invariance.

13 Directed learning and architecture rewrites

Fix input and output interfaces, a dataset $D = \{(x_i, y_i)\}_{i=1}^N$, and a smooth loss ℓ. Define

$$L_D(A, w) = \frac{1}{N} \sum_{i=1}^{N} \ell(F_A(w, x_i), y_i).$$

This is a mathematical objective for a fixed dataset. We make no claim about population risk from this definition. Because it depends only on the represented function on the dataset, it is invariant under every exact circuit symmetry.

Let the training-state stack be

$$S_{\text{train}} = \coprod_{[A]} [\, W_A \times O_A \times M_A \,/ G_A \,].$$

The chosen action on O_A and M_A must transform any coordinate-indexed optimizer quantities. For example, a momentum vector transforms with the model weights. A scalar step-size parameter is invariant. A coordinate-specific learned controller must include its reindexing action rather than silently treating all coordinates as fixed names.

Directed edges over this state object include accepted gradient changes, accepted changes of optimizer parameters, and accepted architecture rewrites. The path higher monad of Section 4 composes them. They are not automatically 2-cells in $\mathcal{N}_{\text{sym}}$, since they generally change the represented function or the available parameter dimension.

13.1 A noninvertible exact widening

For one hidden neuron with contribution $v\,\sigma(a \cdot x + \alpha)$, replace it by two neurons with the same incoming data and contributions

$$\lambda v\,\sigma(a \cdot x + \alpha) \quad \text{and} \quad (1 - \lambda) v\,\sigma(a \cdot x + \alpha).$$

Their sum equals the original contribution for every x and every real λ. The other neurons stay fixed. This supplies a parameter map $e_\lambda : W_A \to W_{A'}$ satisfying

$$F_{A'}(e_\lambda(w), x) = F_A(w, x).$$

The identity is proved directly by distributivity. It is an instance of the function-preserving expansion principle used in network transfer methods [11].

The rewrite increases the parameter space. A general tuple of independently chosen duplicated neurons cannot be recovered by reversing this prescribed duplication map. Thus the widening is a directed rewrite, not automatically an equivalence of parameterized architectures. Equality of the currently represented function does not provide an inverse on all future parameters.

If a particular neuron is selected, its identity must be part of the proposal data. Under a permutation, that selected neuron is reindexed. Alternatively one can widen all neurons symmetrically. This supplies the equivariance needed for the operation to be defined on the quotient stack.

If I includes a parameter-count cap, the rewrite requires a proof that the widened architecture still satisfies that cap. Its loss-change certificate has equality, but its memory cost is positive. Widening by itself therefore establishes representational transfer, not strict objective improvement or a runtime advantage.

13.2 Residual insertion and typed interfaces

Another exact rewrite replaces a circuit f by a residual extension

$$x \longmapsto f(x) + r_\theta(x), \qquad r_0(x) = 0.$$

At initialization $\theta = 0$, the output function is unchanged. The color of this instruction records compatible output dimensions; otherwise addition is not typed. If the new branch shares parameters or optimizer memory with another branch, that dependency belongs in the state type.

The tree signature can use both widening and residual insertion as constructors. Their proofs establish the initial loss relation. Subsequent training in the enlarged architecture still requires fresh analytic certificates. The old gradient dimension, curvature constant, and stationarity condition do not remain valid merely because the initialization preserved the old function.

13.3 Learning steps descend through permutations

Proposition 13.1 (Permutation-equivariant gradient update)**.** *Let a finite group act orthogonally on W_A, and let L_D be differentiable and invariant under that action. Then*

$$\nabla L_D(gw) = g \nabla L_D(w).$$

For an invariant scalar η, the map $w \mapsto w - \eta \nabla L_D(w)$ is equivariant and induces a map of the action groupoid.

Proof. Differentiate $L_D \circ g = L_D$ at w. In gradient notation, $g^\top \nabla L_D(gw) = \nabla L_D(w)$. Orthogonality gives the stated equality. Subtract the equivariant gradient term from gw to obtain equivariance of the update. The equations respect multiplication in the group and therefore define a groupoid map, with its nerve-level coherence. □

An optimizer with momentum has an equally explicit obligation. If $m' = \beta m + \nabla L_D(w)$ and $w' = w - \eta m'$, send (w, m) to (gw, gm). Invariance of β, η makes both formulas equivariant. The same reasoning applies to a learned updater only when its parameter action makes its evaluation map equivariant.

13.4 Why arbitrary reparameterization is different

For an invertible linear coordinate change $v = Cw$, the gradient transforms as

$$\nabla_v (L_D \circ C^{-1})(v) = C^{-\top} \nabla_w L_D(w).$$

This is not $C \nabla_w L_D(w)$ unless the required metric compatibility holds. For a scalar example, take $L(w) = w^2/2$ and $v = cw$. Training w and then changing coordinates gives $v' = (1 - \eta)v$, whereas taking a Euclidean gradient step in the v coordinate gives $v' = (1 - \eta/c^2)v$. These differ for general c.

One repair is to carry a metric as part of the optimizer semantics and transport it under the coordinate change. The gradient vector field of a scalar objective with respect to that transported metric is coordinate natural. A finite update map additionally needs a natural integration or retraction choice; coordinate naturality of a vector field alone does not make arbitrary coordinate-wise Euler formulas natural under nonlinear maps.

This example explains why a higher semantic quotient must be selected together with the operations that are intended to descend through it. Quotienting all parameterizations by current behavioral equality would identify systems whose future learning dynamics differ. We do not impose that quotient on the training process.

14 Recursive improvement of a neural optimizer

We now instantiate genuine optimizer modification. Fix an architecture for the analytic portion. Let $o \in \mathbb{R}^k$ be the parameters of a learned optimizer network and m its memory. Its evaluation proposes a positive scalar step size

$$(\widehat{\eta}, m_{\text{prop}}) = H_o(\phi(w, \nabla L_D(w), m)).$$

The feature map ϕ is selected to be invariant or equivariant under the declared parameter symmetries, as appropriate. A separate learned or symbolic meta-policy proposes $(o_{\text{prop}}, m_{\text{prop}})$. It can use the current o and its memory, so the source of future proposals evolves.

The runner constrains every executed step to a common interval $[\underline{\eta}, \overline{\eta}] \subset (0, 2/M)$ and to the admissible state type. In ideal real arithmetic a scalar clipping map supplies the interval bound. A constraint on w' can instead be supplied as evidence with the proposal. If no certificate is supplied, the runner idles. This description makes no assumption that a certificate exists at every nonstationary state.

Optimizer-only changes can leave w fixed and preserve the current loss. A future gradient change is accepted by its own contract. Updating o therefore does not automatically establish that o_{prop} is better than o as an optimizer. It establishes that the recursively changing proposer remains inside a class with a common step contract.

Theorem 14.1 (A constrained recursively changing neural proposer)**.** *Assume the following data.*

1. *The dataset, loss, admissibility predicate, and parameter symmetry action are fixed.*
2. *The learned evaluation and meta-policy are well-typed and equivariant for that action.*
3. *Every executed nonidle weight change has the gradient form and the bounds in Theorem 11.2.*
4. *Every accepted optimizer or architecture change carries the evidence required by the fixed graph of certified changes.*

Then every finite interpreted program preserves admissibility and nonincrease of L_D, provided its primitive certificates assert nonincrease. Execution descends through the chosen permutation groupoids. On an infinite fixed-architecture run with infinitely many accepted gradient updates, with all intervening steps leaving w unchanged and with the uniform analytic hypotheses, the accepted-update sequence satisfies Theorem 11.2, independently of how the proposer changes its parameters.

Proof. Equivariance makes the primitive operations and their evidence maps defined on the state groupoids. The path-certificate proposition composes admissibility and loss evidence for every finite history. Interpreting the dependent tree language in this certified semantics preserves substitution by Theorem 8.2, or directly by its free interpretation property.

For the analytic claim, changes in o affect the chosen step sizes but not the displayed sufficient-decrease inequality. Apply Theorem 11.2 to the specified fixed-architecture run. The hypotheses explicitly exclude the inference of infinitely many accepted steps from the possibility of idling, and exclude the transfer of a fixed-dimensional convergence argument across unbounded architecture changes. □

14.1 A typed improvement language for this example

Use endpoint-pair colors for compatible training states as in the common-carrier construction of Section 8. The endpoint types retain network boundaries, architectures, and optimizer compatibility. A primitive signature can contain:

1. a nullary idle-history instruction at an admissible state;
2. an equivariant gradient-step instruction with a supplied segment and step-size bound;
3. a function-preserving widening instruction with its parameter-count evidence;
4. an optimizer-parameter update with its typing and symmetry evidence;
5. a finite composition instruction with matching intermediate state types.

Each operation has finitely many program slots. Continuous scalar or matrix parameters label operations; they are not an unbounded number of recursive argument slots. Thus this can be represented by the finite-arity dependent polynomial signature of Section 5.

An outer program can generate an optimizer-update program and then execute it. The free-language metamonad expresses the additional nesting, and its multiplication expands the generated programs coherently. The acceptance interface still interprets every primitive change. There is no theorem that a deeper syntax tree corresponds to a better optimizer.

14.2 Beyond feedforward networks

The interface construction extends to finite unrolled recurrent networks, attention blocks, graph-message-passing blocks, and neural differential-equation solvers when their primitive semantics and typing are declared. The monadic results only require the category and signature hypotheses, so they do not depend on a specific activation formula.

The analytic hypotheses do depend on the example. A recurrent network with unbounded unfolding needs a continuing-run or partiality semantics. A numerical differential-equation solver needs a stated error model. Attention with a smooth softmax has a smooth evaluation on finite real inputs, but this does not supply a global useful curvature bound. Quantized circuits need an ordered discrete or approximate semantics instead of the smooth gradient calculation. The generalization is structural; each numerical guarantee must be re-established for the selected primitive interpretation.

15 Geometric semantics and change of context

The internal formulation is useful when improvement is specified over changing contexts: local data sources, distributed observations, interface parameters, or smooth families of networks. A geometric morphism $f : \mathcal{F} \to \mathcal{E}$ has a left exact, colimit-preserving inverse image $f^* : \mathcal{E} \to \mathcal{F}$. We now state exactly which constructions transport along it.

Theorem 15.1 (Base change of coherent histories)**.** *For every state object S and graph G over $S \times S$, there is a canonical equivalence*

$$f^*\mathsf{Path}_S(G) \simeq \mathsf{Path}_{f^*S}(f^*G).$$

It respects the units, multiplications, and all higher coherence of the path monads, under the induced functor between graph slices.

Proof. Finite limit preservation identifies f^* of a composable string of edges with the corresponding string after base change. Colimit preservation identifies its dependent sum over intermediate states and its coproduct over lengths with the transported ones. Thus it identifies every summand of Formula (1). Edge insertion and path concatenation use exactly these operations. The associative convolution construction is preserved, so its free algebra interpretation identifies the full coherent monad structure. □

For polynomial instructions one must be more careful: a general dependent product is a right adjoint, and an inverse image need not preserve it. Let $P = \Sigma_t \Pi_p \Delta_s$ have finite locally constant discrete input fibers, and let P' be its base-changed polynomial. Finite products commute with inverse image; the local trivializations of p and descent therefore give the required Beck–Chevalley equivalence $f^*P \simeq P'f^*$ on slices.

Theorem 15.2 (Base change of finite-arity recursive programs)**.** *Under the preceding finite-arity assumptions,*

$$f^*(T_P X) \simeq T_{P'}(f^*X).$$

The equivalence is compatible with returned outputs, substitution, and interpretation whenever the primitive interpretations are transported by the same comparison.

Proof. At stage zero both initial chains contain the initial object. If the stages agree at n, preservation of coproducts and the polynomial comparison identify $f^*(X \amalg PY_n)$ with $f^*X \amalg P'f^*Y_n$. Induction identifies every stage and its transition map. Preservation of sequential colimits identifies the colimits. The returned-output maps and primitive constructors are transported identically, so the free universal property gives compatibility with substitution and coherent interpretation. □

15.1 Predicates, descent, and their limits

An admissibility predicate is a subobject $S_I \hookrightarrow S$. Its inverse image is a subobject of f^*S. Certified preservation maps transport by pullback and composition. This observation uses the predicate as supplied data; it does not claim that inverse image preserves arbitrary universal quantification used to define a new predicate from scratch.

For a covering family of contexts, locally specified parameters and evidence form a global object only if their overlaps satisfy descent, including the higher compatibility on multiple overlaps. The stack of neural symmetries retains precisely the relabeling information needed on such overlaps. A list of locally acceptable networks without compatible overlap identifications need not determine one global network.

The two base-change theorems concern syntax and its coherent interpretation. They do not show that empirical estimates remain valid after changing the data distribution, that external computation costs are preserved, or that a numerical optimization result survives every change of context. An objective must itself be transported with its ordered semantics and any analytic assumptions.

The distinction is particularly sharp for a geometric morphism that forgets information. Its inverse image has exact formal properties, but the forward observational process may discard distinctions essential to a learning rule. A statement about preservation by f^* should not be restated as an unrestricted theorem that every information-losing projection preserves optimality.

16 Obstructions and counterexamples

16.1 Coherence does not force progress

Take the identity monad on any category and the identity transition on every state. Every monad law holds, every admissibility predicate is preserved, and the objective stays constant. Therefore monadic coherence and invariant preservation do not imply strict improvement.

Even strict improvement does not imply an attained optimum. On states s_n with $J(s_n) = 1/(n+1)$ and only arrows $s_n \to s_{n+1}$, every selected step improves the objective, but no state attains zero. A completion adding a limiting state changes the state space and may add no executable way of reaching that state in finite time.

16.2 Self-modification can keep a poor policy

Let $s = (w, o)$, let $J(w, o) = w^2$, and let the updater change only o by an invertible renaming. The optimizer representation changes at every step, but w never moves. This is a self-modifying trajectory with no model improvement. A definition that counts any syntactic code change as recursive improvement would conflate two distinct predicates.

A learned optimizer can also be replaced by a worse one while leaving the present model and present loss fixed. An optimizer-only update therefore needs a separate meta-objective if it is to be called an improvement of optimizer quality. One possible meta-objective is expected future loss over a fixed task distribution and fixed execution budget. Its validity then depends on that distribution and budget.

16.3 No universal effective strict improver

The following obstruction is a direct reduction to the undecidability of halting [12]. It concerns effective optimization, not the consistency of a categorical object representing all mathematical candidates.

Proposition 16.1 (Uniform strict-improvement obstruction)**.** *There is no always-terminating algorithm that, given a program for any total computable $q : \mathbb{N} \to \{0, 1\}$ with $q(0) = 1$, returns an n such that $q(n) < 1$ whenever such an n exists.*

Proof. For a machine code e, construct the total computable function

$$q_e(0) = 1, \qquad q_e(n+1) = \begin{cases} 0, & \text{if machine } e \text{ halts within } n \text{ steps,} \\ 1, & \text{otherwise.} \end{cases}$$

Each value is obtained by a bounded simulation, so the program is total uniformly in e. Run the alleged improver and evaluate q_e at its output. If e halts, an improving input exists, so the returned value must be zero. If e never halts, every returned value is one. This decides halting, a contradiction. □

The same proof excludes an always-terminating universal global-minimizer selector for this class. Every q_e has a minimum, so the obstruction does not arise from nonattainment. The available exits are to restrict the problem class, permit partial search, weaken the guarantee, or accept an independently justified oracle. Replacing programs by typed terms in a higher topos does not make this effective obstruction disappear.

16.4 An objective contract is not a generalization theorem

Let the training set consist only of input 0 with target 0. All functions $f_c(x) = cx$ have zero squared training loss there, while their behavior at 1 varies arbitrarily with c. The certified training-loss category can therefore contain changes with unchanged training loss and arbitrarily different off-dataset error. A population-risk theorem must introduce a distribution, statistical assumptions, or another justified specification.

Likewise, a finite collection of successful tests does not establish an arbitrary universal safety predicate. Our preservation results accept mathematical evidence for each primitive specification. They do not replace that evidence by a finite test suite. This manuscript contains no experimental validation section and makes no claim that a finite benchmark establishes the stated universal obligations.

16.5 Changing the checker or the objective

The protected specification can be extended, but doing so requires an explicit translation between old and new contracts. If the system rewrites its acceptance predicate from I to the constantly true predicate, preserving the new predicate says nothing about the old one. If it replaces loss by a constant, monotonicity of the new loss is similarly uninformative.

A justified specification revision can carry an implication $I_{\text{new}} \Rightarrow I_{\text{old}}$ and an order-preserving comparison of objective meanings. Without those maps, a revision belongs to a different problem. The formalism makes this boundary visible through the context parameter K and the interpretation morphisms.

17 The combined recursive-improvement specification

The preceding sections can be assembled into one specification without identifying their distinct carriers. The following theorem states the resulting conditional package.

Theorem 17.1 (Coherent constrained recursive improvement)**.** *Fix a state object S in an infinity-topos, an admissibility predicate I, an ordered objective J, and a graph G of certified primitive changes. Fix a finite-arity dependent improvement signature P and a lawful interpretation of its constructors in certified finite histories. Permit the state to include an editable, well-typed proposer and permit program-valued operation labels through the free-language metamonad. Then:*

1. *every finite interpreted program yields a directed history preserving I and nonincrease of J when those are the primitive contracts;*
2. *substitution of programs and flattening of nested language expansions give coherently equivalent interpreted histories;*
3. *resource costs add under history composition, with budget grades adding accordingly;*
4. *compatible sequential language extensions assemble into one higher monad when their underlying endofunctors preserve sequential colimits;*
5. *the path and finite-arity tree semantics transport along geometric inverse image;*
6. *the quantitative conclusions of Section 11 hold exactly when their additional analytic or probabilistic hypotheses hold for the selected executions.*

Proof. The free path construction and certificate interpretation prove the first assertion. The free tree universal property and the free-language adjunction prove the second. Summation of primitive costs proves the third. Theorem 9.1 proves the fourth. Theorems 15.1 and 15.2 prove the fifth. The last assertion is an application of the independently stated descent and expectation results to the interpreted histories. No step of this proof derives their numerical hypotheses from monadic structure. □

The theorem deliberately permits optimizer modification while retaining a fixed acceptance interpretation. A future extension can permit certified modifications of that interpretation, but it must carry a morphism of contracts rather than merely replace the checker. Similarly, permitting architecture change does not by itself give a common metric in which the entire changing system converges.

Construction	Mathematical result	Additional obligation
Path monad	Coherent finite composition	Primitive evidence and a policy
Tree monad	Typed recursive syntax	Lawful constructor interpretation
Metamonad	Coherent language expansion	A reason that new proposals help
Sequential colimit	One compatible extended monad	Continuity and admissibility closure
Descent inequality	Objective and stationarity bounds	Uniform executed-step hypotheses
Neural quotient	Exact symmetry and naturality	Equivariance of the updater
Geometric inverse image	Transport of stated syntax	Transport of the intended objective

17.1 Four different limits

It is useful to name four constructions explicitly:

$$\mu Y.\,(X \amalg PY), \qquad \operatorname*{colim}_n T_n, \qquad \lim_{n\to\infty} J(s_n), \qquad \lim_{n\to\infty} s_n.$$

They denote an initial recursive datatype, a categorical extension of monads, a numerical loss limit, and a state-space limit. Their existence arguments differ. The first uses initial-chain continuity; the second uses compatibility of monad structures and sequential continuity; the third can follow from monotonicity and boundedness; the fourth needs a topology and additional convergence assumptions. None can be substituted for another in a proof.

17.2 Scope of the proposed theory

The framework treats recursive improvement as controlled evolution of model, procedure, and language inside a fixed interpreted contract. It is compatible with highly expressive learned proposers, nontrivial higher symmetries, and changing architecture. Its guarantees remain conditional and compositional. It does not supply a universally improving policy, certify an actual software checker, or establish that richer syntax increases intelligence.

This scope is a mathematical strength: it locates exactly where each substantive claim enters. A researcher can replace one proposer by another without redoing the algebraic coherence argument, while still being required to justify the numerical contract that the new proposer satisfies. Conversely, a failure of descent is not concealed by the existence of a coherent higher monad.

A A worked nested-program interpretation

For clarity, use the endpoint-pair colors and common history carrier of Section 8. Let $e : s \to t$ be a certified optimizer-only primitive and let $d : t \to u$ be a certified gradient primitive. Include nullary operations meta_e and grad_d of colors (s,t) and (t,u), and a binary operation $\mathsf{seq}_{s,t,u}$ with those two input colors and output color (s,u). The closed term

$$p = \mathsf{seq}_{s,t,u}\big(\mathsf{meta}_e, \mathsf{grad}_d\big) \in (T_P 0)_{(s,u)}$$

has no unresolved variable leaves. An unrelated intermediate state would not satisfy the composition constructor's color requirements. The subscript 0 is the empty graph of variables, not a distinguished runtime state.

The interpretation of p under $T_P 0 \to \mathsf{H}_G 0 = \mathsf{Path}_S G$ is the path (e, d). If e leaves the model fixed, it has objective equality. If d carries sufficient-decrease evidence, the composite has precisely the same numerical decrease as d, together with both admissibility implications.

Now let an outer operation label be the complete program p. An element of T_{T_P} can contain this label as one node. The metamonad multiplication replaces it by its underlying P tree. Interpreting before or after that replacement yields the same path (e, d) up to the supplied coherent comparison. It does not turn one meta-step into an additional, unaccounted improvement.

Finally, let X be a graph of continuation variables and let $q \in (T_P X)_{(u,v)}$. A term in $T_P(T_P X)$ can compose a lifted copy of p with a returned program variable q of that color. The ordinary monad multiplication $T_P^2 \to T_P$ substitutes q at that return. The interpreted path becomes (e, d) concatenated with the interpretation of q, retaining any variables of X. This is the distinction between expanding a program-valued operation label and substituting a program-valued output.

Every numerical and resource certificate follows the same order as execution. If e, d, q have costs c_e, c_d, c_q, the flattened program has cost $c_e + c_d + c_q$. Counting the label p as a cost-free primitive would be a different resource interpretation; coherence of syntax cannot justify that change.

B Explicit obligations for a future mechanization

No Lean source is supplied in this version. The following is a precise boundary for a future substantial formal development, not a claim of completed machine checking. A meaningful neural-network mechanization would include the parameterized circuit category, its permutation groupoids, exact widening, the induced free-history interpretation, and a nontrivial optimizer equivariance theorem. Encoding only the two low-dimensional monad equations on sets would not mechanize the higher semantics developed here.

One possible staged route is to mechanize the strict $(2, 1)$ neural presentation first. Circuit instances become 1-cells, hidden-unit permutations become invertible 2-cells, and typed graph gluing defines composition. The critical checks include interchange, compatibility of gluing with relabeling, and the naturality of the gradient or a specified symbolic derivative. The nerve then relates this presentation to its higher categorical semantics through a stated mathematical bridge.

A stronger development of $\mathcal{N}_{\text{exact}}$ would additionally formalize topological semantic fibers, homotopy quotients, and their composition. The deep-linear gauge example requires retaining the relevant higher topology, so a set quotient cannot substitute for it. An executable optimizer would require a separate numerical representation and error analysis before exact-real proofs could be transferred to machine arithmetic.

The proof obligations of the paper are therefore mathematical statements about explicit structures. A future compiler or runtime should name its trusted kernel, its numerical semantics, its extraction boundary, and its treatment of evidence. None of those implementation properties is assumed from the presence of the words higher monad or dependent type.

References

[1] J. Lurie. *Higher Algebra*. Version of September 18, 2017. See Sections 4.1 and 4.7 for associative algebra and monads. https://www.math.ias.edu/~lurie/papers/HA.pdf.

[2] J. Lurie. *Higher Topos Theory*. Annals of Mathematics Studies 170, Princeton University Press, 2009. https://www.math.ias.edu/~lurie/papers/HTT.pdf.

[3] E. Riehl and D. Verity. Homotopy coherent adjunctions and the formal theory of monads. *Advances in Mathematics* 286 (2016), 802–888. https://arxiv.org/abs/1310.8279.

[4] R. Haugseng. On lax transformations, adjunctions, and monads in $(\infty, 2)$-categories. *Higher Structures* 5(1) (2021), 244–281. https://arxiv.org/abs/2002.01037.

[5] N. Gambino and J. Kock. Polynomial functors and polynomial monads. *Mathematical Proceedings of the Cambridge Philosophical Society* 154(1) (2013), 153–192. https://arxiv.org/abs/0906.4931.

[6] D. Gepner, R. Haugseng, and J. Kock. ∞-Operads as analytic monads. *International Mathematics Research Notices* 2022(16), 12516–12624. See Section 4 for initial algebras and free monads. https://arxiv.org/abs/1712.06469.

[7] C. Rezk. A model for the homotopy theory of homotopy theory. *Transactions of the American Mathematical Society* 353(3) (2001), 973–1007. https://arxiv.org/abs/math/9811037.

[8] B. Fong, D. I. Spivak, and R. Tuyéras. Backprop as functor: A compositional perspective on supervised learning. In *Proceedings of LICS 2019*. https://arxiv.org/abs/1711.10455.

[9] G. S. H. Cruttwell, B. Gavranović, N. Ghani, P. Wilson, and F. Zanasi. Categorical foundations of gradient-based learning. In *Programming Languages and Systems (ESOP 2022)*, 1–28. https://arxiv.org/abs/2103.01931.

[10] M. Andrychowicz, M. Denil, S. Gómez Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems 29*, 2016. https://arxiv.org/abs/1606.04474.

[11] T. Chen, I. Goodfellow, and J. Shlens. Net2Net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*, 2016. https://arxiv.org/abs/1511.05641.

[12] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42(1) (1936–1937), 230–265. https://doi.org/10.1112/plms/s2-42.1.230.