PanLogos for AI: A Modal, Reflective Topos as a Safe Interop Layer for Neuro-Symbolic Systems

Douglas C. Youvan

doug@youvan.com

www.youvan.ai

October 1, 2025

Modern AI stacks mix learned models, discrete programs, and formal tools—yet we lack a single mathematical substrate that makes these parts compose safely. PanLogos offers exactly that: a reflective, modal, univalent $(\infty, 1)$-topos that treats dataflows and proofs as certified diagrams; lets us switch "lenses" (classical, constructive, realizability, sheaf/local) without semantic drift; and turns existence proofs into executable witnesses. In this paper we show how PanLogos immediately facilitates AI capabilities: (i) verified tool-use and planning via safe quotation/evaluation (reflection), (ii) neuro-symbolic bridges where learned maps are arrows in typed diagrams with universal-property checks, (iii) mode-aware reasoning that prevents covert assumption shifts, (iv) diagram-first verification of ML pipelines and refactors, and (v) program extraction from proofs for synthesis and control. We present a reference architecture that embeds LLM agents, optimizers, and sensors as PanLogos objects; define institution-level bridges to proof assistants and set-theoretic notebooks; and report benchmark protocols for round-tripping theorems and pipelines without re-proving. The result is an interoperability and safety layer: prove once, render anywhere, run with guarantees. PanLogos does not replace existing formalisms; it coordinates them, providing compositionality, auditability, and conservative transport—exactly the guardrails modern AI systems need.

Keywords: PanLogos, neuro-symbolic AI, reflective reasoning, tool use, planning, realizability, modal logic, Lawvere–Tierney modality, univalence, diagrammatic reasoning, universal properties, conservativity, proof assistants, program extraction, sheaf models, robustness, interoperability.

A collaboration with GPT-5. CC4.0. 61 pages.

Outline

1. Introduction and Contributions
2. Background: PanLogos in One Page (PL1–PL8)
3. Reference Architecture for AI in PanLogos
   3.1 Agents, tools, and environments as objects and arrows
   3.2 Modes: classical, constructive, realizability, sheaf/local
   3.3 Reflection for planning and tool orchestration
4. Diagram-First Verification of ML Pipelines
   4.1 Typed-graph semantics for dataflows
   4.2 UP-checked refactors and invariants
5. Neuro-Symbolic Bridges without Glue Code
   5.1 Treating learned maps as arrows with contracts
   5.2 Institution bridges to PA libraries and notebooks
6. Executable Witnesses from Proofs
   6.1 Realizability for synthesis and controllers
   6.2 Case: CRT-style model merging with extracted code
7. Mode-Aware Reasoning and Safety
   7.1 Preventing assumption smuggling across modes
   7.2 Sheaf modalities for localization and fusion
8. Evaluation Protocols and Benchmarks
   8.1 Round-trip theorem/pipeline suites
   8.2 Ablations: with vs without reflection/diagram checks
9. Related Work and Positioning
10. Limitations and Risks
11. Roadmap and Conclusion
    Appendices: APIs for bridges, diagram payload format, reproducible examples.

References

1. Introduction and Contributions

Modern AI systems are assemblies of heterogeneous parts: gradient-trained networks, search and planning modules, simulators, formal solvers, data pipelines, and interfaces to human operators. These parts seldom share a single semantics. A model's invariances are informal; a pipeline's refactor may silently change behavior; an agent's plan may rely on an unstated logical principle; a proof artifact from one stack (Lean/Coq/Agda) is awkward to reuse in another. The result is brittle composition and unclear guarantees.

PanLogos offers a unifying host: a reflective, modal, univalent $(\infty, 1)$-topos in which AI artifacts are typed arrows and diagrams with *certified* universal properties (UPs). Modalities act as lenses (classical, constructive, realizability/computation, sheaf/local); reflection supports safe quotation/evaluation of plans and code; bridges provide conservative transport to and from existing formalisms and proof stacks. In this paper we specialize PanLogos to AI practice, showing how it yields immediate benefits for planning, neuro-symbolic interop, program synthesis, verification of ML dataflows, and mode-aware safety.

1.1 Why a topos-shaped host for AI?

- **Compositionality:** Dataflows, controllers, and learned maps become objects/morphisms; refactors are diagram pastings whose UPs can be checked mechanically.
- **Modal discipline:** Mark each step's logic (e.g., classical vs constructive, realizability vs extensional) to prevent covert assumption shifts.
- **Executable content:** Realizability supplies witnesses—turning existence proofs ("there is a decoder/merge operator") into runnable components.
- **Interoperation:** Bridges to proof assistants and set-theoretic notebooks let results travel without re-proving; rendering (graph, TeX, emoji) is separated from semantics.

1.2 Practical pain points addressed

- **Unverified tool orchestration:** Current agents call tools via prompts. In PanLogos, plans are *quoted* programs that are checked as diagrams, then safely evaluated.
- **Pipeline drift:** ETL/feature refactors often change meaning. UP-checked pastings certify that invariants survive refactors.
- **Neuro-symbolic glue:** Ad-hoc wrappers disappear when learned functions are arrows with declared contracts; compositions are verified by UPs.
- **Assumption smuggling:** Classical lemmas cannot silently infect constructive subroutines—mode labels persist throughout derivations and transports.

- **Non-portable proofs:** A lemma proven in Lean can be transported to Coq/Agda or to a set-theoretic notebook via conservative bridges.

1.3 Contributions

1. **Reference Architecture for PanLogos-AI.**
   A typed-graph semantics for agents, tools, environments, and dataflows; modes for classical, constructive, realizability (computational), and sheaf/local reasoning; reflection for safe planning and tool-use.

2. **Diagram-first Verification for ML Pipelines.**
   A library of UP-checkers (products, equalizers, pullbacks/pushouts, (co)limits on finite shapes) to certify ETL and training graphs. We show that common refactors are *provably semantics-preserving* via pasting lemmas.

3. **Neuro-Symbolic Bridges without Glue Code.**
   Treat learned models as arrows with contracts (domain/codomain, invariants); compose with symbolic modules in one diagram. Provide institution-level bridges to import/export proofs and specifications conservatively.

4. **Executable Witnesses from Proofs.**
   Using realizability, extract runnable components (e.g., decoders, merge operators, controllers) from "∃-proofs." Demonstrate on CRT-style model merging and controller synthesis.

5. **Mode-Aware Safety for Reasoning Chains.**
   A discipline that tags each step's logic and ensures left-exact transport across modes; prevents hidden use of classical axioms in constructive subproofs; supports localization via sheaf modalities for multi-sensor fusion.

6. **Evaluation Protocols and Benchmarks.**
   Propose round-trip suites: (i) theorem transport across proof stacks; (ii) pipeline refactor certification; (iii) ablations comparing agents with vs without reflection and diagram checks.

7. **Open-Source Spec and Payload Format.**
   A machine-readable **Diag**payload embedded in exported figures (SVG/PDF) to make explanations auditably tied to certified diagrams; APIs for bridges to popular proof assistants.

1.4 What this paper is not

We do not propose a new proof assistant, a replacement for existing ML frameworks, or a monolithic logic. PanLogos is a *host*: it coordinates existing systems with formal guardrails, enabling "prove once, render anywhere, run with guarantees."

Takeaway. By internalizing diagrams, modalities, and reflection, PanLogos equips AI with a semantics where learned and symbolic parts compose, plans are verified before execution, and results travel across ecosystems without semantic drift.

2. Background: PanLogos in One Page (PL1–PL8)

PanLogos is a reflective, modal, univalent $(\infty, 1)$-topos engineered as a **host** for heterogeneous reasoning. Below, each axiom is stated succinctly with its **AI payoff** and **usage hint**.

PL1 — Univalent universes in an $(\infty, 1)$-topos

What: A cumulative tower $\mathcal{U}_0 \in \mathcal{U}_1 \in \cdots$where identity in $\mathcal{U}_i$coincides with equivalence; higher paths encode "sameness up to structure."
AI payoff: Model-class invariances (two models yield identical predictions) as *equivalences* rather than brittle equalities; transport properties along these equivalences is principled.
Use: Distinguish $=$(definition), $\cong$(isomorphism), $\simeq$(equivalence). Map representation changes to transports.

PL2 — Regular/exact core (images, kernels, effective quotients)

What: Finite limits; image–mono factorizations; kernel pairs with effective coequalizers; exactness (every internal equivalence relation is effective).
AI payoff: ETL/feature pipelines gain categorical guarantees: "well-definedness" = factors through a quotient; stability of images under pullback = refactors preserve invariants.
Use: Encode *checkable* conditions for data cleaning, label quotienting, and model identification.

PL3 — Lawvere–Tierney modalities and modal subtoposes

What: A family of left-exact reflectors $a_\square$: PL $\to$ PL$_\square$(e.g., double-negation/classical, sheaf/local, realizability/computational).
AI payoff: Mark each step's *logic*: classical proofs cannot silently contaminate constructive code;

local reasoning for sensor regions; computational mode extracts runtime artifacts.
Use: Tag proofs and diagrams with □; change lenses via $a_{\square}$without re-proving.

PL4 — Reflection (Metalogos) and conservative evaluation

- **What:** A comonad Mwith quotation t -> ⌜ t ⌝

and evaluation ev: M$X \to X$, commuting with modalities up to iso; **conservative** (no new object-level truths).
AI payoff: Plans = quoted programs; verify against types/diagrams before execution; eliminates "prompt-only" orchestration risks.
Use: Build "propose → check → evaluate" loops; proofs generated in Mmust pass checks before they affect X.

PL5 — Institutional bridges and adjoint conservativity

What: Satisfaction-preserving translations $\mathbf{Br}_{\mathbb{I}}$to/from ETCS/ZFC, HoTT/UF, algebraic specification systems, and proof assistants; with conservative adjoints back.
AI payoff: Move theorems and specs across stacks (Lean/Coq/Agda, notebooks) without semantic drift; reuse verified components.
Use: Treat external artifacts as *first-class*—import, reason, export unchanged.

PL6 — Internal diagram engine (typed graphs, UP checking)

What: An internal category $\mathbf{Diag}$of typed graphs with commutativity constraints and universal-property (UP) predicates (products, equalizers, pullbacks, pushouts, finite (co)limits).
AI payoff: ML/agent pipelines are certified diagrams; refactors are pastings whose semantics are mechanically checked; explanations render from the *same* certified core (graph, TeX, emoji).
Use: Author pipelines as diagrams; attach UP badges; let renders be views, not semantics.

PL7 — Computation/partiality modality and executable content

What: A realizability (or partiality) modality ◊that is left exact; ◊ $\exists x.\, P(x)$corresponds to a runnable witness/program.
AI payoff: From "there exists a decoder/merge operator" to **generated code** with correctness guarantees; certify controller synthesis and search subroutines.
Use: Prove existence under ◊, extract and register the executable component.

PL8 — Conservativity schema

What: Truth is preserved/reflected across bridges and modalities, even with reflection in the loop. Transporting in and out of $PL_{\Box}$does not strengthen statements.
AI payoff: Round-trip safety: a pipeline or lemma moved between stacks returns **unchanged** (modulo explicit tags). Enables reproducible, cross-tool governance.
Use: Make mode/bridge tags part of artifacts; design CI that mechanically checks round-trips.

One-line workflow (for practitioners)

1. **Choose a mode** □(classical / constructive / realizability / sheaf).
2. **Model the system** as a typed diagram $D \in \mathbf{Diag}$; attach UP badges.
3. **Quote plans** in M; **check** diagrammatically; **evaluate** conservatively.
4. **Extract code** via ◊where existence is proven.
5. **Bridge** artifacts to external stacks; **round-trip** to verify PL8.

Net effect: Compositional ML + programs + proofs, with mode-aware guardrails and rendering-independent certification.

3. Reference Architecture for AI in PanLogos

We model an AI stack as a **typed diagram** in PanLogos. Components (agents, tools, datasets, simulators) are **objects**; invocations and dataflows are **arrows**; contracts/invariants are **subobjects**; and rewrites/refactors are **pasting** of universal-property (UP)–certified pieces. Modes label the logical lens; reflection turns plans into quoted programs that are checked before execution.

3.1 Agents, tools, and environments as objects and arrows

3.1.1 Core objects

- **State spaces:** S(world state), B(beliefs), G(goals), H(history), A(actions), O(observations), Θ(parameters).
- **Models/tools:** $f: X \to Y$(e.g., encoder, policy, planner, solver, retriever, verifier).

- **Contracts (predicates/subobjects):** Inv $\hookrightarrow X$(input validity), Spec $\hookrightarrow X \times Y$(I/O relation), Safe $\hookrightarrow A$(safety filter).

3.1.2 Composition and UP-certified nodes

- **Pipelines as products/equalizers:**
  - **Product:** fan-in of features $F = \prod_i \quad F_i$via $\mathrm{UP}_{\times}$.
  - **Equalizer:** enforce agreement, e.g., two parsers $p_1, p_2 : X \to Y$with $\mathrm{Eq}(p_1, p_2) \hookrightarrow X$.
- **Pullbacks (conditioning):** form $X \times_Z Y$to *constrain* a call by a context Z(data joins, conditional plans).
- **Pushouts (merge):** join branches (e.g., speculative decoding) via $\mathrm{UP}_{\mathrm{po}}$.

3.1.3 Agent loop as a diagram

- **Perception:** $e : S \to O$(sensor), $u : O \to B$(update).
- **Deliberation:** $\pi : B \times G \to A$(policy) or $P : B \times G \to$ Plan.
- **Execution:** $\alpha : A \times S \to S$(actuator/simulator).
- **Logging:** $\ell : S \times A \to H$(append-only).
- **Safety gate:** $g : A \to A$with Safe $\hookrightarrow A$and $g(a) = a$on Safe(equalizer characterizes pass-through).

UP-badges certify that these nodes satisfy the expected **universal properties**, so refactors that paste diagrams preserve semantics.

3.1.4 Contracts and certificates

- **Functional contract:** $\mathrm{Spec}_f \subseteq X \times Y$with graph inclusion $\Gamma(f) \subseteq \mathrm{Spec}_f$.
- **Monotonicity/compositionality:** preserved by products/pullbacks in PL2; verified by PL6 over the diagram.
- **Probabilistic wrappers:** wrap a stochastic tool $f$as $X \to \mathrm{Dist}(Y)$; contracts become expectations/credible sets attached as subobjects of $X \times Y$via moment maps.

3.2 Modes: classical, constructive, realizability, sheaf/local

PanLogos reads the same diagram through **modal lenses** $a_{\square}$: PL → PL$_{\square}$(PL3). Mode tags become part of artifacts.

3.2.1 Classical mode ($a_{\neg\neg}$)

- **Use:** offline proofs, crisp set-level invariants, non-constructive existence.
- **Effect:** availability of classical lemmas; outputs are set-valued artifacts.
- **Guardrail:** cannot be imported into constructive code without explicit retagging.

3.2.2 Constructive mode (default/univalent)

- **Use:** certifiable build pipelines, verified properties without LEM.
- **Effect:** ensures computational interpretations exist or are marked as non-constructive if imported from classical.

3.2.3 Realizability / computational mode (◊)

- **Use:** turn "∃ tool/decoder/merge" into a **runnable** witness; controller synthesis.
- **Effect:** proofs ◊ $\exists x.\, P(x)$yield programs; interfaces feed directly into runtime.
- **Guardrail:** executable content is annotation; forgetting ◊does not change classical truths (PL7, PL8).

3.2.4 Sheaf/local mode ($a_{\mathbf{sh}}$)

- **Use:** multi-sensor fusion, federated settings, local validity with overlap gluing.
- **Effect:** objects are *local sections*; consistency is checked on overlaps; descent guarantees are diagram-certified.
- **Guardrail:** global theorems derived from local patches come with explicit *glue* constraints.

3.2.5 Mode choreography

- **Left-exact transport:** $a_{\Diamond} a_{\square} \cong a_{\square} a_{\Diamond}$on finite-limit structure (commutation).
- **Patterns:**
  - prove existence in classical →check computably in ◊→deploy;
  - learn locally (sheaf) →glue →use globally in classical or constructive.

3.3 Reflection for planning and tool orchestration

Reflection (PL4) equips agents with a **safe meta-loop**: *propose* → *check* → *evaluate*. Quotes live in M; evaluation evis conservative.

3.3.1 Planning as quoted programs

- **Plan syntax:** Planis a type of **codes** for compositional calls (tool DAGs).
- **Quotation:** quote: $\mathrm{Plan}_{\mathrm{ext}} \to \mathrm{M}(\mathrm{Plan})$.
- **Evaluation:** $\mathrm{ev}_{\mathrm{Plan}}: \mathrm{M}(\mathrm{Plan}) \to$ Planafter checks.

3.3.2 Checking phase (inside M)

- **Typing:** each tool call $f: X \to Y$is well-typed w.r.t. upstream/downstream objects.
- **Contract satisfaction:** $\Gamma(f) \subseteq \mathrm{Spec}_f$and propagated invariants hold.
- **UP badges:** product/equalizer/pullback nodes satisfy $\mathrm{UP}_\star$.
- **Mode constraints:** every subcall's mode matches its consumers; no implicit classicalization of constructive segments.

If checks pass, a certificate object C(in $\mathbf{Diag}$) is attached to the plan code.

3.3.3 Conservative evaluation and execution

- **Conservativity:** evdoes not create truths—execution only proceeds for plans that had a valid proof object.
- **Runtime guard:** a thin runtime re-verifies cheap invariants (types/domains) to catch drift (e.g., data schema changes).

3.3.4 Orchestrating external tools (bridges, PL5)

- **Bridge-in:** import a Coq/Lean lemma as a node-level contract; import a notebook function as an arrow with declared pre/post-conditions.
- **Bridge-out:** export the certified diagram plus mode tags; external stacks can trust the invariant and re-check locally.
- **Round-trip (PL8):** re-imported artifacts must match; CI can fail the build on mismatch.

3.3.5 Worked micro-patterns

(A) Verified retrieval-augmented generation (RAG).
Objects: Q(queries), $\mathcal{K}$(index), C(contexts), T(texts), A(answers).
Arrows: retrieve $r: Q \to C$, read $d: C \to T$, answer $g: Q \times T \to A$.
Contracts: relevance Rel $\hookrightarrow Q \times C$; faithfulness Faith $\hookrightarrow Q \times T \times A$.
Diagram: product node $Q \times T$, equalizer for consistency checks; UP badges certify the composition.
Modes: $\Diamond$to extract a checker for Faith; classical to carry non-constructive corpus lemmas.

(B) Safe tool chain for code-gen.
Plan codes compose: spec → sketch → synthesize → verify → execute.
The verify node is an equalizer between "run tests" and "prove spec," ensuring only artifacts that agree pass; evaluation allowed only if the equalizer UP holds.

(C) Sheaf-based sensor fusion.
Local maps $s_i: U_i \to O_i$; overlap constraints $O_i \restriction_{U_i \cap U_j} \cong O_j \restriction_{U_i \cap U_j}$.
Pushouts glue local sections; descent predicate certified in $\mathbf{Diag}$. Mode a_{sh}ensures locality; classical shadow yields a global fused estimate.

3.3.6 Minimal checklist for deployable agents

- All nodes/arrows typed; domains/codomains declared.
- Contracts as subobjects; inclusion proofs attached.
- All structural nodes carry UP badges; pastings certified.
- Every step labeled with a mode; no illicit cross-mode calls.
- Plans created in M; checks pass; evaluation logs the certificate.
- Bridges for external components defined; round-trip test green.

Outcome: Agents plan and compose tools **with proofs attached**. Refactors become diagram pastings that *cannot* silently change semantics; existence proofs yield runnable components; and cross-stack movement is conservative by construction.

4. Diagram-First Verification of ML Pipelines

We treat ML pipelines as **typed graphs** whose nodes/arrows are certified by **universal-property (UP)** predicates. Refactors are **pasting** operations that preserve semantics by construction. This yields checkable guarantees for ETL, feature engineering, training/eval, and deployment.

4.1 Typed-graph semantics for dataflows

4.1.1 Basic typing

- **Objects (types).** D(raw data), C(cleaned), F(features), Θ(parameters), M(models), E(embeddings), P(predictions), L(labels), R(residuals), S(scores/metrics), H(artifacts/logs).
- **Arrows (maps).** $c: D \to C$(clean), $f: C \to F$(featurize), $t: (F \times L) \to \Theta$(train), $b: (\Theta \times F) \to M$(bind/build), $p: (M \times F) \to P$(predict), $\ell: (P \times L) \to S$(loss/metric), $u: (S \times \Theta) \to \Theta$(update), $z: F \to E$(embed), etc.

Typing is enforced by PL6's internal category $\mathbf{Diag}$. Composition $g \circ f$ is admissible iff codomain(domain) matches.

4.1.2 Structural nodes with UP badges

- **Product (fan-in).** $F = F_1 \times \cdots \times F_k$ with projections π_i.
  UP $_{\times}$**:** for any Z and maps $h_i: Z \to F_i$, there exists a unique $u: Z \to F$ with $\pi_i \circ u = h_i$.
- **Equalizer (agreement).** $\mathrm{Eq}(g, h) \hookrightarrow X$ where $g, h: X \to Y$.
  UP $_{=}$**:** selects values where two computations agree (e.g., two validators).
- **Pullback (conditioning/join).** $X \times_Z Y$ for $x: X \to Z$, $y: Y \to Z$.
  UP $_{\mathrm{pb}}$**:** used for schema joins, conditioning on context, aligning views.
- **Pushout (merge).** $X \sqcup_Z Y$ for fusing branches (e.g., speculative decoding).
  UP $_{\mathrm{po}}$ guarantees the universal merge.

These UP badges are *checked* internally (PL6). Renderings (graphviz, TeX, emoji) are just views.

4.1.3 Contracts as subobjects

- **Schema contract.** $\mathrm{Sch}_C \hookrightarrow C$(post-clean schema), $\Gamma(c) \subseteq \mathrm{Sch}_C$.
- **Train/serve spec.** $\mathrm{Spec}_p \hookrightarrow (M \times F) \times P$(e.g., monotone w.r.t. threshold).
- **Metric monotonicity.** $\mathrm{Mono} \hookrightarrow S \times S$(score should not worsen after safe transform).

- **Safety.** Safe $\hookrightarrow P$(output policy constraints).

Contracts are preserved by products/pullbacks (PL2), so composing certified arrows preserves invariants.

4.1.4 Probabilistic/stochastic wrappers

Represent stochastic components as $X \to \mathrm{Dist}(Y)$. Attach contracts as *moment* or *credal* subobjects:

- mean map $\mu\colon \mathrm{Dist}(Y) \to Y$,
- credible region $\mathrm{CR} \hookrightarrow X \times Y$with $\Pr[(x,y) \in \mathrm{CR}] \geq 1-\delta$.
  UP checks proceed on the deterministic skeleton; contracts record the probabilistic side conditions.

4.1.5 Example: training–serving loop

Objects: C, F, Θ, M, P, L, S.
Arrows: $c\colon D \to C$, $f\colon C \to F$, $t\colon (F \times L) \to \Theta$, $b\colon (\Theta \times F) \to M$, $p\colon (M \times F) \to P$, $\ell\colon (P \times L) \to S$, $u\colon (S \times \Theta) \to \Theta$.
Structure:

- product nodes $F \times L$, $\Theta \times F$, $P \times L$;
- equalizer node $\mathrm{Eq}(p \circ (b \circ (t, f)), \tilde{p})$to ensure offline vs online predictors agree;
- pullback to align new $F$with historical Lon a join key.

All structure nodes carry UP badges; contracts assert schema validity, metric dominance, and safety.

4.2 UP-checked refactors and invariants

Refactors are **diagram pastings** that replace a subdiagram by a UP-equivalent one. Because UPs uniquely characterize the object/arrow "up to unique iso," the overall semantics is preserved.

4.2.1 Canonical refactor patterns

(R1) Feature split/merge (product associativity/commutativity).
Replace $(F_1 \times F_2) \times F_3$with $F_1 \times (F_2 \times F_3)$or reorder features.

- **Guarantee:** product UP implies unique mediators; pipelines remain isomorphic.

- **CI check:** confirm mediating isos are used only via projections; no order-sensitive code downstream.

(R2) Validator consolidation (equalizer factorization).
Two-step check $X \overset{g,h}{\to} Y$ then $Y \overset{g',h'}{\to} Z$ refactored into a single equalizer at X w.r.t. composites $g' \circ g, h' \circ h$.

- **Guarantee:** equalizer of composites pulls back along equalizer inclusion; uniqueness preserves "pass" sets.

(R3) Contextualization (pullback introduction).
Instead of manual filtering on keys, form $X \times_Z Y$.

- **Guarantee:** UP $_{\text{pb}}$ yields the initial object making both constraints hold; avoids silent data leakage.

(R4) Branch fusion (pushout).
Speculative branches B_1, B_2 with shared boundary Z merged as $B_1 \sqcup_Z B_2$.

- **Guarantee:** UP $_{\text{po}}$ ensures any consistent downstream consumer factors uniquely.

(R5) Precomputation hoisting (Fubini-like).
Swap order of independent products/pullbacks when comparison map is an iso (Beck–Chevalley).

- **Guarantee:** PL6 pasting check + PL2 pullback stability.

(R6) Metric gate reordering (equalizer as guard).
Move an acceptance test earlier if it depends only on already-available data.

- **Guarantee:** equalizer's mono property ensures no new passes, possibly fewer (safe conservative optimization).

4.2.2 Invariant schemas that survive refactors

- **Schema invariants:** Sch $\hookrightarrow C, F$ preserved by products and pullbacks; verified once per node.
- **Monotone post-processing:** if $q: P \to P$ is monotone and p respects Safe, then $q \circ p$ respects Safe.
- **Calibration/consistency:** equalizer-based alignment guarantees p matches a reference predictor on a certified subset.

- **Provenance:** pushout merges propagate provenance edges; uniqueness gives a canonical lineage.

4.2.3 Soundness lemmas (pipeline-level)

- **Product invariance lemma.** If downstream consumers use only projections, then replacing a product node by any UP $_{\times}$-isomorphic product preserves outputs pointwise.
- **Pullback correctness lemma.** A join implemented as a pullback equals the set of pairs satisfying both predicates; manual filters refactored to pullbacks cannot add spurious pairs.
- **Equalizer guard lemma.** Moving an equalizer earlier in the pipeline is refinement: accepted set shrinks or stays; no false positives are introduced.

Each lemma is a one-line consequence of the respective UP plus mono/epi facts (PL2, PL6).

4.2.4 CI integration (practical checklist)

- **Schema checkers:** attach $\Gamma(f) \subseteq$ Schproofs to ETL nodes; fail build if a refactor lacks a proof.
- **UP certification:** require UP badges for product/equalizer/pullback/pushout nodes; re-run PL6 checks on every change.
- **Refactor lints:** maintain a library of (R1–R6) rewrites with automatically generated mediators; fail on ad hoc rewires.
- **Mode discipline:** tag test-time assertions as classical; training checks as constructive or ◊when extracting code.
- **Round-trip tests:** export the certified diagram payload with the artifact; re-import after deployment; fail if mismatch (PL8).

4.2.5 Worked micro-examples

(E1) Safer join.
Old: filter → hash-join (hand-coded).
New: pullback $C \times_K L$on key K.

- **Proof:** UP $_{\mathrm{pb}}$ + equality of composites shows equivalence; removes class of join bugs.

(E2) Feature reorder.
Old: concatenate $[f_2, f_1, f_3]$. New: $[f_1, f_2, f_3]$with a permutation iso before model.

- **Proof:** product symmetry; add the mediating iso; downstream uses projections so semantics unchanged.

(E3) Gate hoist.
Old: compute expensive $p$then gate with Safe.
New: gate earlier using only inputs required by Safe.

- **Proof:** equalizer mono + factorization; early failure reduces cost without accepting any previously-rejected items.

Takeaway

With PL6's diagram engine and PL2's exact/regular core, ML pipelines can be **proved correct by structure**: products/equalizers/pullbacks/pushouts characterize the key nodes; refactors are certified pastings; invariants ride along as subobjects. This makes ETL, training, and serving **rendering-independent**, **mode-aware**, and **CI-checkable**—a practical route to robust, evolvable ML systems.

5. Neuro-Symbolic Bridges without Glue Code

PanLogos removes the ad-hoc "glue layer" by (i) treating learned components as **arrows with contracts** inside a certified diagram, and (ii) providing **institution-style bridges** to proof assistants (PAs) and computational notebooks that preserve satisfaction and reflect truth (PL5, PL8).

5.1 Treating learned maps as arrows with contracts

5.1.1 Arrow schema for learned components

A trained model is an arrow $f: X \to Y$with attached **contracts** (subobjects) describing guarantees and expectations:

- **Domain/Schema contract:** $\mathrm{Dom}_f \hookrightarrow X$(e.g., tokenization, shape, normalization).
- **Postcondition contract:** $\mathrm{Post}_f \hookrightarrow X \times Y$(e.g., Lipschitz bound, monotonicity, calibration).
- **Uncertainty:** $f: X \to \mathrm{Dist}(Y)$with mean $\mu$and credible set $\mathrm{CR}_f \hookrightarrow X \times Y$.

- **Safety filter:** Safe $\hookrightarrow Y$ with a projector $g: Y \to Y$ satisfying $\Gamma(g) \subseteq$ Safe and $g\,|_{\text{Safe}} = \mathrm{id}$.

Registration: include inclusions $\Gamma(f) \subseteq \text{Post}_f$ and $\mathrm{im}(x \mapsto (x, f(x))) \subseteq \text{Dom}_f \times Y$. These are first-class proofs in the diagram (PL6).

5.1.2 Composition without glue

Contracts compose *by structure*:

- **Sequential:** if $f: X \to Y$ respects Post_f and $g: Y \to Z$ respects Post_g, then $g \circ f$ respects the **pullback** contract

$$(\mathrm{id}_X \times g)_{\backslash *}\ \text{Post}_f \ \wedge\ (X \times \Gamma(g))^{\backslash *}\ \text{Post}_g,$$

checked via UP $_{\mathrm{pb}}$ (PL2, PL6).

- **Parallel (fan-in):** for product $\langle f_1, \ldots, f_k \rangle: X \to \prod_i Y_i$, the contract is the **product** $\bigwedge_i (\text{Post}_{f_i})$ with UP $_{\times}$.
- **Agreement gates:** equalizers enforce "consistency" between redundant modules (e.g., two parsers), ensuring downstream consumers see only agreed outputs.

Result: no bespoke adapter code; the categorical laws propagate contracts automatically.

5.1.3 Training and evaluation as certified subdiagrams

- **Empirical contracts:** learned bounds (e.g., calibration) are registered as subobjects together with *evidence morphisms* (datasets $\mathcal{D}$ give arrows $\mathcal{D} \to X \times Y$).
- **Generalization hooks:** theoretical bounds (e.g., PAC) appear as classical-mode lemmas linked to empirical subobjects, keeping mode labels explicit (PL3).

5.1.4 Spec-to-model alignment

A *symbolic* spec $S \hookrightarrow X \times Y$ and a learned f align by proving $\Gamma(f) \subseteq S$ or by exhibiting a **mediator** $m: X \to Y$ such that $\Gamma(m) \subseteq S$ and an **equalizer** forces f to agree with m on a certified subset. This yields **refinement** diagrams instead of bespoke wrappers.

5.1.5 Worked micro-patterns

- **Calibrated classifier:** $f: X \to \text{Dist}(\{0,1\})$. Contracts: $\text{Cal}_\epsilon \hookrightarrow X \times [0,1]$ s.t. $|\ \mathbb{E}[Y \mid p] - p\ | \leq \epsilon$. A **link** $p \mapsto \mathbf{1}_{p>\tau}$ preserves a monotonicity contract; safety is enforced by an equalizer with a policy constraint.

- **Retriever with coverage:** $r: Q \to \mathcal{P}(C)$. Contract: coverage $\text{Cov}_\alpha \hookrightarrow Q \times C$ ensuring recall $\geq 1 - \alpha$. A reader d composes via product; agreement with a verifier v enforced by equalizer.

5.2 Institution bridges to PA libraries and notebooks

PanLogos integrates external formal/compute stacks via **institutional bridges** (PL5): satisfaction-preserving translations with conservative adjoints back (PL8). This removes glue code between proofs, notebooks, and runtime.

5.2.1 Targets and roles

- **Proof assistants (Lean/Coq/Agda):** import lemmas/specs as contracts; export PanLogos-certified diagrams back as lemmas with witnesses or as tactic scripts.
- **Notebooks / runtime (Python/Julia/Rust):** import functions as arrows with types/contracts; export certified plans for execution; keep a round-tripable diagram payload.

5.2.2 Bridge schema (ETCS/HoTT/spec → PanLogos)

Given an external artifact $\mathcal{A}$:

1. **Syntactic map:** $\text{Sig}(\mathcal{A})$ ↦internal types/objects; predicates ↦subobjects; equations ↦equalities in PL.
2. **Semantic map:** models/implementations ↦arrows f with graph $\Gamma(f) \subseteq \text{Post}_f$.
3. **Truth layer:** if $\mathcal{A}$ proves ϕ, then $\mathbf{Br}(\mathcal{A})$ satisfies the corresponding internal statement; adjoint back reflects it unchanged (PL8).

5.2.3 Bridge adapters (pragmatic API sketch)

- **From PA lemma to contract**
  - *Input:* PA theorem ∀x. P(x) → Q(f x).
  - *Bridge:* produce a subobject $\text{Post}_f := \{(x, y) \mid P(x) \Rightarrow Q(y)\} \hookrightarrow X \times Y$ and an inclusion proof $\Gamma(f) \subseteq \text{Post}_f$.
  - *Mode:* classical/constructive per PA; label with $a_{\square}$.
- **From PanLogos diagram to PA lemma**
  - *Input:* certified diagram with UP badges and inclusions.

- *Bridge:* generate PA goals: UP statements as existence/uniqueness lemmas; inclusions as pointwise implications; produce proof terms/tactics guided by the diagram structure.
- *Conservativity:* resulting PA theorem equals the PanLogos statement modulo explicit tags.

- **From notebook function to arrow**
  - *Input:* Python function def f(x): ....
  - *Bridge:* assign types X, Y, derive schema contract from type checker/tests, import as arrow $f: X \to Y$ plus evidence morphisms from test datasets; optionally attach runtime monitor proofs (lightweight).
- **Back to notebook/runtime**
  - *Input:* certified plan (quoted, checked) with witnesses from ◊.
  - *Bridge:* emit an executable graph (e.g., DAG) whose nodes correspond to arrows; include the **diagram payload** (contracts, UP badges) for audit and re-check.

5.2.4 Mode-aware transport

- **Classical → constructive:** allowed only with explicit tags (no silent LEM import). If a PA lemma is classical, PanLogos enforces a classical mode on consumers or requires a constructive reproof.
- **Realizability witnesses:** when exporting to runtime, attach executable artifacts produced under ◊; when re-importing to a PA, they become existence proofs with extracted code.

5.2.5 Round-trip guarantees (PL8)

- **No strengthening:** a lemma/spec imported from Coq and then exported back must remain identical (up to view/mode annotations).
- **Diagram fidelity:** exported SVG/PDF carries the machine-readable **Diag**payload; re-import reconstructs the same internal object.

5.2.6 Worked bridge scenarios

- **(B1) Lean lemma → safer post-processor.**
  Lean proves: "if $p$calibrated and $q$monotone, then $q \circ p$preserves safety." Bridge produces Post$_{q \circ p}$and inclusion proofs; PanLogos uses it to certify a new serving graph without additional glue.

- **(B2) Coq spec → Python component.**
  A Coq refiner r with spec Spec_r is imported as arrow $r: X \to Y$; PanLogos composes it with learned f, verifies contracts, and exports to a Python DAG with the spec embedded and checkable.
- **(B3) Notebook model → PA certification.**
  A PyTorch model f is imported with empirical calibration evidence; a PanLogos diagram enforces an equalizer against a spec model on a certified subset. Exported to Lean as a lemma: "on subset S, f agrees with spec," with the subset and agreement produced from the diagram.

Takeaway

By making learned components **first-class arrows with contracts** and by using **institutional bridges** to import/export truths conservatively, PanLogos eliminates brittle glue code. Composition, refactoring, and verification are handled by **universal properties** and **mode-aware transport**—so neuro and symbolic parts truly interoperate, with proofs (and runnable artifacts) attached.

6. Executable Witnesses from Proofs

In PanLogos, the realizability/partiality modality ◊(PL7) turns "there exists x" statements into **runnable artifacts**. Practically: you prove a synthesis or control theorem under ◊, and PanLogos yields an executable witness (code or circuit) whose I/O contract is the very proposition you proved. PL8 ensures that forgetting ◊*never* changes the classical truth you started with.

6.1 Realizability for synthesis and controllers

6.1.1 Contract-to-witness pattern

- **Spec as subobject.** A requirement $P(x)$ lives as Spec $\hookrightarrow X$.
- **∃-claim under ◊.** A derivation of $\Diamond\ \exists x{:}X.\, P(x)$ establishes *constructibility*.
- **Extraction.** By PL7, the proof corresponds to a partial (or total) program wit: $\mathbf{1} \rightharpoonup X$ with proof $\Gamma(\text{wit}) \subseteq \text{Spec}$.
- **Registration.** wit is attached to the diagram node; its correctness certificate *is* the inclusion proof.

6.1.2 Synthesis library (typical schemata)

- **Decoder/encoder pairs.** Show ◊ $\exists d.\, d \circ e = \mathrm{id}$on a certified subset; extract d(e.g., invertible preprocessor on its image).
- **Merge/align operators.** Given overlaps $U_i \cap U_j$with compatibility, prove ◊ $\exists$ glue: $\prod_i \quad O_i \to O$satisfying descent; extract glue.
- **Controller synthesis.** For dynamics $s_{t+1} = F(s_t, a_t)$and safety Safe $\hookrightarrow S$, prove ◊ $\exists \pi: S \to A$s.t. $F(s, \pi(s)) \in$ Safe; extract π.

6.1.3 Controller example (sketch)

Spec. Barrier function $B: S \to \mathbb{R}$, threshold τ, and action set $A$with order $\preceq$.
Goal. ◊ $\exists \pi.\, \forall s \in$ Safe, $B \qquad (F(s, \pi(s))) \leq B(s)$and $F(s, \pi(s)) \in$ Safe.
Witness. Extract $\pi(s) = \arg\min_{a \in A} B \qquad (F(s, a))$with a fallback to the previous safe action (encoded as a realizability choice operator).
Guarantee. The inclusion proof shows the closed-loop arrow $S \overset{\langle \pi, \mathrm{id} \rangle}{\to} A \times S \overset{F}{\to} S$factors through Safe $\hookrightarrow S$.

6.1.4 Partiality and timeouts

If the synthesis is inherently partial (e.g., may fail on corner cases), witis a partial map with **domain certificate** $D \hookrightarrow \mathbf{1}$or $D \hookrightarrow$inputs. At runtime, the agent guards evaluation by checking membership in D(cheap predicate carried by the diagram).

6.2 Case: CRT-style model merging with extracted code

The Chinese Remainder Theorem (CRT) yields a canonical **merge operator**. In PanLogos, its regular proof (PL2, PL6) plus ◊produces *runnable merging code* that respects a formal contract.

6.2.1 Problem setup (AI-flavored)

- Two experts give predictions $u: R/I$and $v: R/J$(e.g., heads specializing on complementary feature sets), with ideals $I, J \trianglelefteq R$and $I + J = R$.
- We want a single prediction $w: R/(I \cap J)$and a **merge** $\mathrm{merge}: R/I \times R/J \to R/(I \cap J)$such that projecting back recovers u, v.

Contract (subobject).

$$\text{CRTSpec} \hookrightarrow (R/I \times R/J) \times R/(I \cap J)$$

requiring $\pi_I(w) = u$and $\pi_J(w) = v$.

6.2.2 Regular proof (structure only)

- Build $\Phi: R \to R/I \times R/J$with kernel $I \cap J$.
- By PL2, factor through the quotient to $\bar{\Phi}: R/(I \cap J) \xrightarrow{\cong} R/I \times R/J$.
- Inverse Ψexists by UP/iso checks (PL6). Define $\text{merge}(u, v) := \Psi(u, v)$. This proves **existence and uniqueness** classically.

6.2.3 Realizability witnesses (executable merge)

To execute merge, we need *constructive ingredients*:

- Find $e_I, e_J \in R$with $e_I \equiv 1 (\text{mod} I)$, $e_I \equiv 0 (\text{mod} J)$and $e_J \equiv 0 (\text{mod} I)$, $e_J \equiv 1 (\text{mod} J)$, plus $e_I + e_J = 1$.
- Then

$$\text{merge}(u, v) = u \cdot e_J \ + \ v \cdot e_I \quad \in R/(I \cap J).$$

◊-extraction. Under standard computational hypotheses (e.g., $R = \mathbb{Z}$or a PID; ideals given by generators), the proof includes:

- an extended-GCD routine (or ideal membership solver) that computes $e_I, e_J$from generators of I, J;
- a function that lifts residues $u, v$to representatives and reduces the linear combination modulo $I \cap J$.

PanLogos records these as the **witness program** merge_codewith the inclusion $\Gamma(\text{merge\_code}) \subseteq \text{CRTSpec}$.

6.2.4 Typed I/O for deployment (copy-paste spec)

- **Types:**
  IdealGen := List[R] (generators),
  ResidueI := R mod I, ResidueJ := R mod J,
  ResidueIJ := R mod (I∩J).

- **Arrow:**
  merge : (ResidueI × ResidueJ × IdealGen(I) × IdealGen(J)) → ResidueIJ.
- **Contracts:**
  projI ∘ merge = fst, projJ ∘ merge = snd (checked via equalizer UP);
  merge total if extended-GCD succeeds; domain certificate otherwise.

6.2.5 Guarantees by design

- **Correctness:** immediate from the diagram's iso and the inclusion proof (PL6, PL2).
- **Mode transparency:** classical existence tagged $\square_{\neg\neg}$; executable path tagged ◊. Forgetting ◊leaves the classical CRT isomorphism unchanged (PL8).
- **Composability:** merge is just an arrow; it composes contract-first with other modules (e.g., decoders, safety gates) without glue code.

6.2.6 Variants

- **n-way CRT.** Inductively merge via pushouts/products, extracting a tree of witnesses; UPs ensure associativity up to unique iso.
- **Sheafified merging.** If Rvaries over a base space (local models), compute merges on overlaps and glue with a sheaf modality; realizability extracts local witnesses, while descent certifies the global arrow.

Takeaway

With ◊, PanLogos turns "there exists a component with property P" into **code with a proof attached**. For synthesis and control, this yields deployable modules whose semantics is guaranteed by the very theorem that produced them; for CRT-style merging, it produces a drop-in operator that is correct-by-construction and composes in larger, UP-certified diagrams.

7. Mode-Aware Reasoning and Safety

PanLogos enforces **mode discipline**: every statement, subdiagram, and artifact carries an explicit *mode tag* (classical $\neg\neg$, constructive, realizability ◊, sheaf/local a_{sh}, …). Left-exact reflectors transport facts between modes with no strengthening (PL3, PL8). This prevents covert assumption shifts ("assumption smuggling") and makes localization/fusion principled.

7.1 Preventing assumption smuggling across modes

7.1.1 What can go wrong (outside PanLogos)

- A constructive routine silently invokes a lemma proved using excluded middle (LEM).
- A runtime module is assumed total because a proof exists—ignoring that the proof lived in a *partiality* semantics.
- A global claim is made from locally valid heuristics with no overlap checks.

7.1.2 PanLogos discipline (rules of engagement)

- **R1 (Explicit tags).** Each node/lemma $X$is annotated by a mode $a_{\square}$. CI fails if a consumer in mode $\square'$uses Xwithout an authorized transport $a_{\square'} \circ j_{\square}$.
- **R2 (Left-exact transport only).** Use only the provided reflectors $a_{\square}$(PL3). No ad-hoc coercions.
- **R3 (No classicalization by accident).** Moving from constructive to classical requires $a_{\neg\neg}$; the reverse requires either a constructive proof or an explicit "shadow" lemma that is known to descend.
- **R4 (Realizability boundaries).** A ◊-proof yields code/witnesses; forgetting ◊erases executables but does not assert totality/classical existence beyond what was proved (PL7, PL8).
- **R5 (Audit trails).** Mode transitions are recorded on the diagram edges; exported artifacts embed the trail in the diagram payload.

7.1.3 Enforcement mechanisms

- **Static checkers.** A mode type-checker traverses $\mathbf{Diag}$: every incoming edge to a node must either (i) match the node's mode or (ii) be the image of a permitted reflector.
- **Witness polarity.** If a consumer expects executable content, the upstream proof must reside in ◊; if not, CI warns about missing executable witnesses.
- **Shadowing lemmas.** Libraries include "descent" theorems: e.g., *HoTT equivalence ⇒ classical isomorphism under 0-truncation*. The checker uses these to justify constructive use of classical results.

7.1.4 Patterns

(M1) Classical existence → executable witness.
Prove $\exists x.\, P(x)$in $a_{\neg\neg}$. Use a separate constructive derivation under ◊(or a realizability

extraction lemma) to produce wit. The runtime path depends only on wit; the classical path remains documentation.

(M2) Constructive subroutine inside a classical proof.
Allow $a_{\neg\neg}$-level reasoning to call a constructive subdiagram (safe monotone). The reverse is disallowed unless the classical claim is known to descend (checker enforces).

(M3) Verified fallback.
If a ◊-witness is partial, compose with an equalizer guard and a classical fallback lemma that is tagged separately. The composition is safe because the guard is constructive and the fallback path is mode-isolated.

7.1.5 Minimal checklist (CI)

- Every node has a mode tag.
- All cross-mode edges pass through an $a_{\Box}$reflector (or a registered descent lemma).
- Executable requirements sourced only from ◊-proofs.
- No constructive consumer depends on a purely classical premise.
- Mode trail embedded in the exported diagram payload; round-trip verified (PL8).

7.2 Sheaf modalities for localization and fusion

Sheaf modalities model **local validity**: propositions hold on *patches* with certified **overlap consistency**; global results arise by **gluing** (descent). This captures multi-sensor fusion, federated learning, and geo/temporal locality.

7.2.1 Setup

- **Cover.** A family $\{U_i \to U\}$of "regions" (spatial, temporal, task, client).
- **Local objects.** $X_i := X \restriction_{U_i}$with restriction maps to overlaps $U_{ij} := U_i \cap U_j$.
- **Sheaf reflector.** a_{sh}: PL → $\mathrm{PL}_{\mathrm{sh}}$sends objects to their "locally checked" counterparts; left exact (PL3).

7.2.2 Local reasoning primitives

- **Sections:** $\Gamma(U_i, X)$= local models/parameters/claims.
- **Compatibility:** equalizers on overlaps enforce $x_i \restriction_{U_{ij}} = x_j \restriction_{U_{ij}}$.

- **Descent (UP).** A global $x \in \Gamma(U, X)$exists iff the family $(x_i)$is compatible and satisfies the sheaf UP predicate (certified in $\mathbf{Diag}$).

7.2.3 Fusion as universal property

- **Pullbacks for alignment.** Join heterogeneous local data via pullbacks on shared keys/coordinates.
- **Pushouts for merge.** Combine complementary modalities along a common interface.
- **Gluing map.** A canonical glue: $\prod_i \quad \Gamma(U_i, X) \to \Gamma(U, X)$defined when compatibility equalizers hold; uniqueness by UP.

7.2.4 AI patterns

(S1) Multi-sensor fusion.
Local estimators $e_i : S \mid_{U_i} \to O_i$. On overlaps, enforce $O_i \mid_{U_{ij}} \cong O_j \mid_{U_{ij}}$(equalizers). UP-certified pushouts yield a fused estimate $O$with provenance; a_{sh}records locality.

(S2) Federated learning.
Clients $U_i$train local θ_i. Overlaps encode common data distributions or shared constraints. A *sheaf aggregator* aggis extracted under ◊to produce θsuch that restrictions $\theta \mid_{U_i}$agree with $\theta_i$on certified subsets. Gluing is justified by descent; non-IID drift is detected as overlap inconsistency.

(S3) Map stitching / SLAM.
Local maps $m_i$with relative transforms on overlaps; equalizers enforce frame consistency; pushouts glue maps; uncertainties tracked via credal subobjects. The global map is the descent object.

7.2.5 Safety properties "for free"

- **No hallucinated global claims.** Without compatible overlaps, the descent predicate fails—no global section is produced.
- **Localized failure.** Inconsistency is pinpointed to overlap constraints; global pipeline remains well-typed/guarded.
- **Composability with modes.** You can reason locally under ◊(extract local witnesses) and only classicalize the final summary if desired; left exactness guarantees stability.

7.2.6 Minimal design recipe

1. Choose a cover $\{U_i\}$and define restrictions.
2. Prove local statements (possibly with ◊witnesses).

3. Encode overlap equalities via equalizers in $\mathbf{Diag}$.
4. Apply $a_{\mathrm{sh}}$and certify the descent UP.
5. (Optionally) bridge out: export fused artifacts plus the diagram payload; consumers can re-check overlaps.

Takeaway

Modes are not labels for show—they are **guardrails**. With explicit tags and left-exact reflectors, PanLogos blocks assumption smuggling and makes locality first-class: *prove locally, glue globally,* and keep executable vs. non-executable content honest. This yields safer reasoning chains and robust fusion in real AI systems.

8. Evaluation Protocols and Benchmarks

We propose **repeatable**, **tool-agnostic** evaluations that stress the two promises of PanLogos for AI: (i) *truth and semantics transport unchanged* (PL5, PL8), and (ii) *structure-first verification prevents silent drift* (PL2, PL6, PL4). Each protocol specifies tasks, artifacts, metrics, and pass/fail gates suitable for CI.

8.1 Round-trip theorem/pipeline suites

8.1.1 Theorem round-trip (formal libraries)

Goal. A lemma proven in stack A (e.g., Lean) is transported into PanLogos, optionally transformed (mode shift, diagram certification, reflection-normalization), and exported to stack B (e.g., Coq) **without strengthening or weakening**.

Suite T-10 (regular/exact core).

- T1 First Isomorphism (Grp/Ring/Mod)
- T2 Image–mono factorization stability under pullback
- T3 Equalizer of product ≅ product of equalizers (finite family)
- T4 CRT (two ideals)
- T5 Beck–Chevalley for pullbacks
- T6 Correspondence theorem for subgroups

- T7 Exactness: every internal ER is effective
- T8 Kernel–cokernel pair identities in Ab
- T9 Regular epi pullback stability
- T10 Finite-limit sketch instance (two models are isomorphic)

Artifacts.

- Source proof term(s) (Lean/Coq) and statement.
- Bridge map **Br**json (signature, sentences).
- PanLogos certificate (diagram payload + mode tags).
- Exported target proof obligations/tactics.

Metrics.

- **RT-eq (strict):** theorem text equality up to alpha-renaming and explicit mode tags.
- **RT-prov:** target stack proves exported obligations *without human edits*.
- **PL-norm:** PanLogos normalization (diagram + reflection) does not alter logical content (hash of normalized statement).
- **Time:** end-to-end latency (import → export → check).
- **Footprint:** size of payload (KB) and proof term length.

Pass criteria. All T1–T10 achieve RT-eq, RT-prov; no unsanctioned mode jumps detected; footprint within 2× of source proof size.

8.1.2 Pipeline round-trip (ML dataflows)

Goal. An ML pipeline specified as a certified diagram runs in a notebook/runtime; the executed artifact and logs are re-imported and re-validated **bit-for-bit** against the original certification.

Suite P-8 (ETL→Train→Serve).

- P1 ETL with schema evolution (one breaking, one non-breaking refactor)
- P2 Feature split/merge (product re-association)
- P3 Safer join (manual filter → pullback)
- P4 Gate hoist (equalizer moved earlier)

- P5 Spec alignment (equalizer against a reference model)
- P6 Metric monotonicity under post-processor
- P7 A/B rollout: pushout branch merge with provenance
- P8 Serving drift: detection via contract violation replay

Artifacts.

- Canonical diagram (.diag JSON) with UP badges and contracts.
- Runtime DAG (e.g., Python/Rust) auto-emitted from the diagram.
- Execution log with "witness hashes" for key nodes (inputs/outputs summarized deterministically).

Metrics.

- **SemEq:** outputs identical on certified subsets after refactor (hash equality or epsilon-bounded numeric delta with proof).
- **UP-ok:** all UP checks pass before and after refactor.
- **Contr-ok:** all inclusions $\Gamma(f) \subseteq \text{Spec}_f$ re-verified from logs.
- **Prov:** lineage/provenance paths preserved (graph isomorphism on provenance subgraph).
- **DriftDet:** when we inject controlled schema or distribution drift, the earliest failing contract is flagged.

Pass criteria. P1–P7: SemEq, UP-ok, Contr-ok, Prov all true. P8: DriftDet pinpoints first violated inclusion or UP.

8.1.3 HoTT⇄Classical descent suite

Goal. Equivalence-level theorems in a univalent region descend to classical isomorphisms **only** under explicit truncation hypotheses.

Suite H-5.

- H1 $A \times B \simeq B \times A \Rightarrow$ classical $A \times B \cong B \times A$
- H2 Universal property of products: equivalence of presentations ⇒ unique iso
- H3 Free object equivalence ⇒ classical adjunction isomorphisms

- H4 Group completion equivalence that **fails** to descend without 0-truncation (negative test)
- H5 HIT-based spec requiring explicit descent lemma (boundary test)

Metrics.

- **Descent-ok:** exported classical statement exists iff truncation tag present.
- **No-smuggle:** constructive consumers never depend on purely classical steps.

8.2 Ablations: with vs without reflection/diagram checks

We isolate the contributions of PL4 (reflection) and PL6 (diagram engine).

8.2.1 Agent planning ablation (Reflection)

Task. Tool-using agent (retrieval + code-run + verifier) on a benchmark like program repair or answers-with-citations.

Conditions.

- **R-on:** plans are quoted, type/contract-checked, then evaluated.
- **R-off:** agent composes tools by prompts only (no quotation, no static checks).

Metrics.

- **PlanValid:** fraction of plans passing type/contract checks (R-on only).
- **ExecFail:** runtime tool errors (HTTP 4xx/5xx, type mismatch).
- **SpecViol:** violations of declared contracts (e.g., missing citation, unsafe action).
- **TaskSucc:** final task success rate.
- **Waste:** tokens/time wasted on invalid/irreversible branches.

Expected outcome. R-on lowers ExecFail and SpecViol significantly and increases TaskSucc with modest overhead.

8.2.2 Pipeline refactor ablation (Diagram checks)

Task. Apply (R1–R6) refactors to a mid-size pipeline (e.g., 20–50 nodes).

Conditions.

- **D-on:** every structural node carries a UP badge; CI blocks merges without re-verification.
- **D-off:** same refactors applied manually (best-effort developer care).

Metrics.

- **SemEq:** post-refactor equivalence on certified subsets.
- **BugRate:** number of seeded defects escaping into evaluation (wrong join, wrong feature order, broken gate).
- **FixCost:** edits/time to repair to pass SemEq.
- **CI-Block:** count of blocked PRs that would have introduced a bug.

Expected outcome. D-on reduces BugRate to near zero and flags CI-Block pre-merge; D-off shows non-trivial defect leakage.

8.2.3 Mode discipline ablation

Task. Mixed pipeline with classical lemmas and constructive runtime components.

Conditions.

- **M-on:** mode tags enforced; cross-mode calls require reflectors or descent lemmas.
- **M-off:** no mode enforcement.

Metrics.

- **Smuggle:** number of illicit classical assumptions used by constructive subroutines (static analysis).
- **Crash/Undefined:** runtime failures due to non-constructive premises.
- **Repro:** round-trip consistency of exported artifacts.

Expected outcome. M-on eliminates Smuggle; improves Repro; M-off shows sporadic crashes or silent assumption shifts.

8.2.4 Realizability extraction ablation

Task. Prove "∃ decoder/merge/controller" and deploy extracted witness vs a hand-coded version.

Conditions.

- **Z-on:** witness extracted under ◊, registered as arrow with inclusion proof.
- **Z-off:** handwritten implementation from a spec document.

Metrics.

- **SpecSat:** measured conformance to Specon held-out tests.
- **DriftRes:** robustness under distribution shift (kept contracts).
- **BugSurfaced:** number of counterexamples found by property-based testing.
- **Perf:** runtime latency/throughput.

Expected outcome. Z-on dominates SpecSat and DriftRes; Perf within tolerances; Z-off occasionally violates edge-case contracts.

8.2.5 Reporting template (for all ablations)

- **Setup:** stacks, versions, seeds, dataset hash.
- **Diagram payload:** SHA of .diag JSON; list of UP badges; mode map.
- **Results table:** metrics ± 95% CI over N runs.
- **Counterexamples:** minimal failing subdiagrams (automatically exported).
- **Cost:** wall-clock/time-to-green-CI, human edit minutes.

Reproducibility & CI Hooks

- **Single source of truth:** store the **diagram payload** (types, UPs, contracts, mode trail) as an artifact; all scripts derive DAGs and tests from it.
- **Deterministic hashing:** hash of (diagram, bridge maps, witnesses) must match across import/export.

- **Auto-oracles:** generate oracle outputs on **certified subsets** only; reject comparisons outside those sets.
- **Seeded drift:** inject controlled schema changes and distribution shifts; ensure earliest failing contract is recorded.

Success bar (what counts as "works")

- ≥ 9/10 theorems in T-10 round-trip with RT-eq and RT-prov.
- ≥ 7/8 pipelines in P-8 pass SemEq/UP-ok/Contr-ok; P8 flags earliest failure.
- Ablations show statistically significant gains ($p < 0.01$) in TaskSucc (R-on vs R-off) and reductions in BugRate (D-on vs D-off).
- All exported artifacts re-import with identical hashes (PL8), and no mode violations recorded.

Bottom line: These protocols let teams *measure* PanLogos's promises—transport without drift, structure-first correctness, safe planning, and executable witnesses—using the same diagrams they will ship to production.

9. Related Work and Positioning

PanLogos sits at the intersection of **neuro-symbolic AI**, **formal methods**, and **category-theoretic/compositional semantics**. It does not attempt to replace mature tools (proof assistants, ML frameworks, workflow engines); instead it supplies a *host* with three distinguishing properties: (i) **modal lenses** (classical/constructive/realizability/sheaf) with left-exact transport, (ii) **reflection with conservativity** (quote–check–evaluate without creating new theorems), and (iii) **diagram-internalized universal properties** that make refactors semantics-preserving and rendering-independent.

9.1 Neuro-symbolic and differentiable programming

Neuro-symbolic systems (logic+NN hybrids, theorem-guided search, differentiable interpreters) typically hard-code interfaces between learned and discrete modules. Differentiable programming (JAX/PyTorch) offers compositional *numerical* semantics, not logical contracts. **Positioning.** PanLogos treats learned maps as *arrows with contracts* and composes them via products/equalizers/pullbacks, so safety and invariants travel structurally. Modal tags prevent

classical lemmas from leaking into constructive runtimes; realizability yields executable witnesses from proofs—capabilities not native to standard neuro-symbolic stacks.

9.2 Formal methods and proof assistants

Proof assistants (Coq/Lean/Agda/Isabelle) provide trusted kernels but live in distinct foundations and file formats. **Program logics** (Hoare/Separation Logic), **model checking**, and **contract systems** certify components but rarely give cross-tool *transport without drift*.
Positioning. PanLogos supplies **institutional bridges** with conservative adjoints: results round-trip between assistants or notebooks **unchanged** (up to explicit mode tags). Reflection is *deflationary*: code generation and tactic scripting occur inside PanLogos yet cannot mint new object-level truths.

9.3 Category-theoretic frameworks

Lawvere theories, sketches, classifying toposes model algebraic structure; **string-diagram calculi** (symmetric monoidal, ZX-calculus) deliver pictorial proof systems; **open systems** frameworks (structured cospans, decorated cospans, lenses/optics) support compositional engineering and cyber-physical design.
Positioning. PanLogos **hosts** these, adds univalence (distinguishing $=$, $\cong$, $\simeq$), and internalizes diagrams with UP-checkers so that proofs are *graph-certified* and **rendering-independent** (TeX, string, emoji). Modalities let the same diagram be read classically, constructively, computationally, or locally—without re-proving.

9.4 Workflow/orchestration and data lineage

Pipelines (Airflow, Dagster, TFX, MLMD) manage DAGs, metadata, and lineage but do not prove that refactors preserve semantics. **DB schema tools** enforce types, not categorical invariants.
Positioning. PanLogos gives a **semantic layer** above orchestration: products/equalizers/pullbacks/pushouts with UP badges certify joins, gates, merges, and re-associations. CI can block merges that violate structure, and exported DAGs carry a machine-readable *diagram payload* for re-checking.

9.5 Probabilistic programming and Bayesian ML

PPLs (Stan, Pyro, Turing) offer measure-theoretic semantics and inference automation; some support effect handlers and verification of samplers.
Positioning. PanLogos wraps stochastic components as arrows $X \to \mathrm{Dist}(Y)$with **credal**/moment contracts that compose by pullbacks/products. It governs *how stochastic pieces glue to symbolic ones* and how their guarantees persist across refactors and mode changes.

9.6 Sheaf-theoretic ML, federated learning, and SLAM

Recent work uses **sheaves** for multi-view learning, sensor fusion, and non-IID federated training; **topological data analysis** adds geometric invariants.
Positioning. PanLogos packages these ideas as a **sheaf modality**: local proofs/witnesses on patches, equalizer checks on overlaps, and descent-style gluing. Failures localize to overlap constraints; global claims require certified compatibility.

9.7 Program extraction and realizability

Classical **realizability** and **proof-to-program** techniques exist (e.g., Coq's extraction, partiality monads). They are stack-specific and may weaken guarantees in translation.
Positioning. PL7 defines a *left-exact* realizability lens ◊inside the same host that carries your diagrams and bridges. "There exists a decoder/merge/controller" becomes **runnable code** with the inclusion proof $\Gamma(\text{code}) \subseteq \text{Spec}$attached; forgetting ◊does not alter classical truths (PL8).

9.8 Safety, governance, and interpretability

Spec-driven agents, **toolformer** approaches, **causal graphs**, and **policy gates** improve reliability but lack a unifying semantics.
Positioning. Reflection (PL4) gives a *propose → check → evaluate* loop; diagram certificates and mode trails become governance artifacts that survive export/import. Interpretations render from the same certified object—no divergence between graph, TeX, or emoji views.

9.9 What PanLogos is *not* (and how it fits)

- **Not a new proof assistant.** It bridges and coordinates existing ones.
- **Not an ML framework.** It sits *above* PyTorch/JAX to certify structure and contracts, then emits DAGs.
- **Not just another diagram language.** Diagrams have *semantics* (typed graphs with UPs), not only syntax.
- **Not a monolithic foundation.** It is **pluralist**: multiple modes and external foundations live inside via conservative bridges.

9.10 Snapshot comparison

Dimension	**Mainstream neuro-symbolic**	**Proof assistants**	**Workflow/DAG tools**	**Sheaf ML**	**PanLogos**
Compositional semantics	Ad-hoc glue	Strong, siloed	Operational only	Local/gluing focus	**UP-certified diagrams**
Cross-stack transport	Minimal	Weak (exports only)	N/A	N/A	**Conservative bridges (PL5/PL8)**
Mode discipline	None	Per-stack	None	Local only	**Modal lenses (PL3)**
Program extraction	Task-specific	Yes (stack-bound)	No	Sometimes	**Realizability ◊(PL7)**
Reflection safety	Prompt-based	Meta tactics (unsafe by default)	N/A	N/A	**Conservative reflection (PL4)**
Rendering independence	No	No	No	No	**Yes (PL6)**

Positioning statement

Use PanLogos when you need *foundation-agnostic* proofs attached to AI pipelines; when refactors must be semantics-preserving by construction; when agents must plan under *checked* quotations; when you want executable witnesses from existence proofs; or when moving artifacts across theorem provers/notebooks without semantic drift. **Keep your current tools**—PanLogos is the **interoperability and certification layer** that lets them work together with explicit modes and universal-property guarantees.

10. Limitations and Risks

PanLogos introduces a powerful—but opinionated—abstraction layer. Below we make the boundaries explicit and list concrete risks with suggested mitigations.

10.1 Theoretical and Foundational Limits

- **Partial completeness only.** Guarantees are strongest for *regular/exact* mathematics and finite-diagram UPs. Higher-order features (power objects, full higher inductives) or essentially higher-homotopical arguments may not descend to classical fragments.
  Mitigation: tag such results with mode/truncation metadata; avoid silent projection to strict isomorphisms.
- **Modal commutation is not automatic.** Distinct reflectors (e.g., double-negation vs. realizability vs. sheaf) need coherence proofs; without them, cross-mode pipelines can be brittle.
  Mitigation: restrict to a verified sublattice of commuting modes; require explicit comparison maps when composing modalities.
- **Universe/size issues.** Exact completions and sheafification can increase universe levels; careless lifting can break univalence or smallness assumptions.
  Mitigation: enforce level accounting in CI; document any “universe bumps” in exported payloads.

10.2 Diagram Engine and Proof Artefacts

- **Finite-shape bias.** PL6 focuses on finite typed graphs; very large or dynamic topologies (online A/B grids, adaptive graphs) stress checkers.

Mitigation: certify reusable motifs (macros) and compose them; isolate high-churn zones behind stable interfaces.

- **UP mis-specification.** A wrong UP badge can "prove the wrong thing" cleanly.
  Mitigation: require local derivations for each badge (existence + uniqueness) and randomize subgoal audits; treat UPs as code-reviewed proofs, not labels.
- **Render/encoding drift.** Emoji/TeX/graph views can diverge visually across platforms.
  Mitigation: ship a canonical machine-readable diagram payload; treat views as lossy projections.

10.3 Bridges and Interop

- **Conservativity assumptions.** Bridges rely on satisfaction-preserving translations; mismatches in implicit axioms (choice, classicality) cause subtle drift.
  Mitigation: mode-tag every imported lemma; block constructive consumers from classical premises unless a descent lemma exists.
- **Granularity mismatch.** Proof assistants expose fine-grained terms; notebooks expose coarse code cells. Round-tripping can inflate artifacts or obscure provenance.
  Mitigation: normalize at the *diagram* level; hash subdiagrams; attach provenance edges explicitly.

10.4 Realizability and Executable Witnesses

- **Partiality at runtime.** Extracted witnesses may be partial (timeouts, non-termination on edge cases).
  Mitigation: require domain certificates; guard evaluation with equalizers/gates; provide verified fallbacks.
- **Spec–world gap.** Correct-by-proof code can still fail due to unmet environment assumptions (APIs, data drift).
  Mitigation: encode operational assumptions as contracts; re-check inclusions against live logs; fail closed.

10.5 Performance and Scalability

- **Checker overhead.** UP verification and mode audits add latency in CI and possibly at deploy.

Mitigation: cache subproofs; incrementally re-check only changed subgraphs; pin canonical normal forms.

- **Large models as arrows.** Treating huge neural nets as single arrows hides internal structure that might matter for optimization.
  Mitigation: expose summaries (Lipschitz bounds, calibration) as contracts; introduce internal substructure only when it pays.

10.6 Usability and Process

- **Authoring burden.** Writing contracts and UP-structured diagrams is extra work.
  Mitigation: provide templates for common motifs (RAG, ETL joins, feature merges, gates); generate scaffolds from code and logs.
- **False sense of security.** Passing checks can be over-trusted; PanLogos validates *structure and declared contracts*, not unspecified behaviors.
  Mitigation: mandate explicit "out-of-scope" notes; pair structural checks with statistical tests where relevant.

10.7 Security and Governance

- **Reflection misuse.** Quotation/evaluation can be tricked into executing unintended code if admission checks are weak.
  Mitigation: keep evaluation deflationary; require plan certificates; sandbox runtime; separate data from control channels.
- **Supply-chain risks.** Imported artifacts (lemmas, code) may be malicious or stale.
  Mitigation: require signed payloads; verify hashes of diagrams, bridges, and witnesses; enforce version pinning.

10.8 Evaluation Threats to Validity

- **Benchmark gaming.** Round-trip suites can be overfit (teaching to the test).
  Mitigation: rotate hidden variants; include negative tests (non-descent cases, failing overlaps).
- **Metric myopia.** Exact graph isomorphism or hash equality can miss meaningful semantic deviations (e.g., nondeterministic tolerances).

Mitigation: pair structural metrics with domain-specific acceptance regions and proofs of equivalence up to those regions.

10.9 Ethical/Operational Considerations

- **Opaque contracts.** If contracts are authored by a small group, others may lack visibility or recourse.
  Mitigation: make contracts first-class, reviewable artifacts; attach rationale and provenance; open change proposals.
- **Locality vs fairness.** Sheaf-style localization may entrench local biases if overlap constraints are weak.
  Mitigation: audit overlaps for representational coverage; encode fairness constraints as contracts that must glue globally.

10.10 Clear Boundaries of Scope

- PanLogos **does not** guarantee statistical generalization, privacy, or fairness by itself.
- It **does not** replace formal verification of low-level implementations (e.g., memory safety).
- It **does not** make undecidable problems decidable; it disciplines *what is checked* and *how artifacts travel*.

Summary

PanLogos trades raw flexibility for **certified structure and explicit modes**. The main risks are mis-specified contracts, over-reliance on badges, and cross-mode/bridge drift. The countermeasures are procedural (reviews, CI audits, provenance) and technical (left-exact reflectors, normal forms, domain certificates). Used with these guardrails, PanLogos can raise assurance without claiming guarantees it cannot deliver.

11. Roadmap and Conclusion

PanLogos is intentionally pragmatic: start with the fragments that already buy reliability in day-to-day AI work (regular/exact structure, finite diagrams, reflection discipline), then widen the aperture. The roadmap below lists concrete artifacts and pass/fail gates so teams can adopt incrementally.

11.1 Roadmap

Phase A — Core substrate ("PanLogos v1")

Goals. Ship the minimal host that makes pipelines certifiable and plans safe.
Deliverables.

- **Diagengine** for finite typed graphs with UP checkers (products, equalizers, pullbacks, pushouts, small (co)limits).
- **Mode kernel (PL3)** with two lenses: classical $a_{\neg\neg}$and realizability ◊(left-exact; comparison maps).
- **Reflection core (PL4)**: quote–check–evaluate loop with deflationary evaluation; CI hook that rejects uncertified plans.
- **Exact/regular library (PL2):** images, kernel pairs, quotients; factorization lemmas packaged as reusable motifs.
  Exit criteria. Pass the P-8 pipeline suite and R-on ablation wins (Sec. 8).

Phase B — Bridges and round-trip ("Interop Alpha")

Goals. Transport theorems and specs across stacks without drift.
Deliverables.

- **ETCS/Lean bridge** for regular/exact fragment; **HoTT bridge** for 0-truncated consequences; **Notebook bridge** (Python DAG ↔ diagram payload).
- **Exporter/importer** that embeds machine-readable diagram payloads in SVG/PDF and reconstitutes them bit-for-bit.
  Exit criteria. ≥ 9/10 items in T-10 (Sec. 8) achieve RT-eq and RT-prov; exported artifacts re-import with identical hashes (PL8).

Phase C — Executable witnesses ("Synthesis Beta")

Goals. Turn existence proofs into deployable code paths.
Deliverables.

- ◊**extraction kit** for CRT-style merge, decoder inversion on image, barrier-based controller.
- **Runtime guards** (domain certificates, equalizer gates) and verified fallbacks.
  Exit criteria. Z-on ablation dominates Z-off in SpecSat/DriftRes; performance within agreed envelope.

Phase D — Sheaf/local and multi-agent ("Locality & Fusion")

Goals. Make locality first-class and gluing certified.
Deliverables.

- **Sheaf modality** $a_{\mathrm{sh}}$with overlap/equalizer tooling; descent predicate and diagnostics.
- **Federated & SLAM motifs** (covers, overlap checkers, gluing maps) as reusable patterns.
  Exit criteria. S1–S3 case studies reproduce; failures localize to overlaps; no hallucinated globals.

Phase E — Coherence & scale ("Reliability Hardening")

Goals. Raise assurance and throughput.
Deliverables.

- **Modal commutation proofs** (comparison cells, Beck–Chevalley squares) for the deployed lens set.
- **Incremental checkers** and cacheable normal forms for large diagrams; provenance hashing for subgraphs.
- **Universe/size ledger** documenting level bumps; automated level checks in CI.
  Exit criteria. Mode-discipline ablation eliminates smuggling; CI stays green under heavy churn.

Phase F — Community & ecosystem

Goals. Make PanLogos a shared medium rather than a silo.
Deliverables.

- **Spec and reference implementation** under a permissive license; test corpora for T-10, P-8, H-5.

- **Adapters** for at least two proof assistants and two workflow engines.
- **Guides**: authoring contracts, writing UP-motivated refactors, adding modes/bridges.
  Exit criteria. External teams reproduce benchmark passes; third-party motifs appear in the library.

11.2 Adoption playbook (for teams)

1. **Start small:** cast one critical pipeline as a $\mathbf{Diag}$with contracts; add UP badges; wire CI to block uncertified merges.
2. **Tag modes:** mark classical vs constructive segments; isolate runtime expectations behind ◊witnesses.
3. **Bridge one way:** import a few lemmas/specs from your PA; use them as contracts; defer export until stable.
4. **Turn on reflection:** require plans to compile under quote–check–evaluate before execution.
5. **Round-trip:** once stable, export diagrams back to your PA or docs with embedded payload; verify hashes.

11.3 Risks we accept en route

- **Scope restraint:** early phases focus on regular/exact + finite diagrams; higher phenomena arrive later with explicit tags.
- **Process overhead:** authoring contracts and badges adds friction; templates and generators must pay down the cost.
- **Cultural fit:** proof and ML communities have different cadences; bridges must respect both.

11.4 Conclusion

PanLogos reframes AI engineering as **diagram-first mathematics with executable witnesses and explicit modes**. By hosting classical, constructive, computational, and local lenses in one univalent topos—and by coupling a reflection discipline with institutional bridges—PanLogos delivers three concrete benefits:

- **Compositional correctness:** refactors are pastings of UP-certified pieces; invariants ride along structurally.
- **Safe interoperability:** theorems, specs, and pipelines move across tools without semantic drift; exported artifacts carry their own proofs.
- **Actionable proofs:** existence claims yield runnable code with correctness certificates; forgetting executables never changes classical truth.

This is not another monolith. It is a **medium** for pluralistic mathematics to meet modern AI practice—where proofs, programs, and pipelines share a single object, can be rendered in many views, and travel conservatively between ecosystems. If the community builds the core fragments, proves the commuting squares, and populates the library of motifs, PanLogos can make robust, auditable AI not an exception but a default.

Appendices: APIs for Bridges, Diagram Payload Format, Reproducible Examples

These appendices are implementation-oriented. Everything is copy/paste-safe and designed to drop into a reference repo.

A. Bridge APIs (Lean/Coq/Agda ↔ PanLogos, Notebooks/Runtime ↔ PanLogos)

A.1 Common concepts

- **ArtifactID:** stable sha256 of canonicalized JSON (see A.4); used for caching and round-trip checks.
- **ModeTag:** one of constructive, classical, realizability, sheaf/local.
- **ProofRef:** external pointer (e.g., PA file+symbol) or inline proof term hash.

A.1.1 Error codes (shared)

- E_MODE: disallowed cross-mode import.
- E_SIZE: universe/level bound exceeded.
- E_SYN: syntactic mismatch (signature/arity).
- E_SEM: failed satisfaction check.
- E_UP: missing or invalid UP badge.
- E_CONTR: contract inclusion proof missing.
- E_HASH: round-trip hash mismatch.

A.2 Proof Assistant Bridge (example: Lean ↔ PanLogos)

A.2.1 Import theorem → Contract/Diagram

Endpoint: POST /bridge/lean/import
Input (JSON):

```
{
  "source": {"tool": "lean", "version": "4.12.0", "file": "Iso1.lean", "symbol": "Group.first_iso"},
  "assumed_axioms": ["choice?no", "LEM?yes"],
  "mode": "classical",
```

```
 "signature": {
  "objects": ["G","H"],
  "arrows": [{"name":"f","dom":"G","cod":"H"}]
 },
 "statement": "∃ (ϕ : G/ker f ≅ im f), true"
}
```


Output (JSON):


```
{
 "artifact_id": "sha256:...",
 "mode": "classical",
 "contracts": [
  {
   "name": "FirstIso",
   "over": ["G","H","f"],
   "type": "isomorphism",
   "payload": {"lhs": "G/ker(f)", "rhs": "im(f)"}
  }
 ],
 "diagram_patch": { "...": "UP badges for kernel pair, coeq, image factorization" },
 "proof_ref": {"tool": "lean", "symbol": "Group.first_iso", "hash": "sha256:..."}
}
```


A.2.2 Export diagram/contract → Proof obligations

Endpoint: POST /bridge/lean/export
Input: Diagram payload (see B) plus list of obligations (UP, inclusions).
Output: Lean file snippet with tactics or exact proofs; artifact_id recorded.

A.3 Notebook/Runtime Bridge (Python/JAX/PyTorch ↔ PanLogos)

A.3.1 Import function → Arrow+Contracts

Endpoint: POST /bridge/nb/import
Input (JSON):

```
{
  "source": {"tool":"python","module":"mymodel","symbol":"predict","hash":"sha256:..."},
  "types": {"dom": "FeatureVec[d=512]", "cod": "ProbVec[k=3]"},
  "contracts": [
    {"name":"Calib_eps","kind":"calibration","epsilon":0.05},
    {"name":"Safe","kind":"range","interval":[0.0,1.0]}
  ],
  "evidence": {"dataset_hash":"sha256:...","tests_passed":1234}
}
```

Output (JSON):

```
{
  "artifact_id": "sha256:...",
  "arrow": {"name":"predict","dom":"F512","cod":"P3"},
  "contract_inclusions": [
    {"contract":"Calib_eps","status":"proven_or_empirical","proofref":"sha256:..."},
    {"contract":"Safe","status":"checked"}
  ]
}
```

A.3.2 Export plan → Executable DAG

Endpoint: POST /bridge/nb/export
Input: Certified plan (quoted, checked) + realizability witnesses.
Output:

- dag.json (nodes/edges with image references),

- runner.py (generated adapter),
- embedded diagram_payload.json with the same artifact_id.

A.4 Canonicalization & Hashing (for reproducibility)

- **Key ordering:** lexicographic keys, arrays sorted by name where order is immaterial.
- **Whitespace:** none; UTF-8; newline \n.
- **Numbers:** decimal, no trailing zeros; NaNs not allowed in payload.
- **Hash:** sha256 over canonical JSON bytes; record as sha256:<hex>.

B. Diagram Payload Format (canonical JSON)

B.1 Top-level schema


```
{
  "version": "1.0",
  "mode": "constructive",
  "universe_level": 0,
  "objects": [
    {"id":"F","type":"FeatureVec","meta":{"dim":512}},
    {"id":"P","type":"ProbVec","meta":{"k":3}}
  ],
  "arrows": [

{"id":"predict","dom":"F","cod":"P","impl":{"ref":"python:mymodel.predict","hash":"sha256:..."
}}
  ],
  "contracts": [
    {"id":"SafeP","over":["P"],"kind":"range","payload":{"min":0,"max":1}},
    {"id":"Calib","over":["F","P"],"kind":"calibration","payload":{"epsilon":0.05}}
  ],
  "inclusions": [
    {"id":"Incl1","graph_of":"predict","into":"SafeP","proofref":"sha256:..."}
  ],
  "structure": {
    "nodes": [
      {"id":"Prod1","kind":"product","legs":["F","P"],"badge":{"UP_times":true}},
      {"id":"Eq1","kind":"equalizer","legs":["p1","p2"],"badge":{"UP_eq":true}},
      {"id":"PB1","kind":"pullback","legs":["x: X→Z","y: Y→Z"],"badge":{"UP_pb":true}},
```

{"id":"PO1","kind":"pushout","legs":["i: Z→X","j: Z→Y"],"badge":{"UP_po":true}}

],

"edges": [

{"src":"F","dst":"Prod1","role":"π1"},

{"src":"P","dst":"Prod1","role":"π2"}

]

},

"mode_trail": [

{"step":0,"mode":"constructive"},

{"step":5,"mode":"realizability","via":"a_diamond"}

],

"witnesses": [

{"id":"merge_code","kind":"realizability","ref":"python:crt.merge","hash":"sha256:...","contracts":["CRTSpec"]}

],

"provenance": [

{"id":"src_lean","kind":"theorem","tool":"lean","symbol":"Group.first_iso","hash":"sha256:..."}

],

"artifact_id": "sha256:COMPUTED"

}

B.2 Minimal required fields

- version, mode, objects[], arrows[], artifact_id.
- For any structure.nodes[*].kind in {product,equalizer,pullback,pushout} you **must** include a badge with UP_*: true.
- Any executable artifact must appear in witnesses[] with kind:"realizability".

B.3 Contracts & inclusions

- **Contract kinds (extensible):** range, calibration, lipschitz, monotone, schema, spec (generic subobject).
- **Inclusion proof:** reference either an external proof (proofref) or an internal derivation (derivation: inlined minimal certificate).

Example inclusion with derivation:

```
{
  "id":"Incl2",
  "graph_of":"safe_post",
  "into":"SafeP",
  "derivation": {
    "uses": ["Eq1","PB1"],
    "sketch": "equalizer mono + pullback stability"
  }
}
```

B.4 Mode trail semantics

- Each mode_trail entry records a *change lens*. The checker verifies that every cross-mode edge is explained by a permitted reflector (e.g., a_negneg, a_diamond, a_sh).

C. Reproducible Examples

C.1 Theorem Round-Trip: First Isomorphism (Grp)

Step 1 — Import from Lean

POST /bridge/lean/import


```
{
  "source":{"tool":"lean","file":"Iso1.lean","symbol":"Group.first_iso"},
  "assumed_axioms":["LEM?yes"],
  "mode":"classical",
  "signature":{"objects":["G","H"],"arrows":[{"name":"f","dom":"G","cod":"H"}]},
  "statement":"∃ ϕ : G/ker f ≅ im f, true"
}
```


Result: artifact_id = sha256:AAA...

Step 2 — Build PanLogos diagram

- Add kernel pair node (UP), coequalizer (UP), image factorization.
- Attach contract IsoSpec representing “G/ker f ≅ im f”.

Step 3 — Export to Coq

POST /bridge/coq/export


```
{
  "artifact_id":"sha256:AAA...",
  "obligations":[
    {"type":"UP_eq","node":"KerPair"},
    {"type":"UP_coeq","node":"Quot"},
    {"type":"iso","lhs":"Quot(G,ker f)","rhs":"im(f)"}
  ]
}
```

Pass criteria: Coq script emitted; proof checks; exported statement equals source up to mode tag classical.

C.2 Pipeline Refactor: Pullback Join

Original: Hand-coded filter+hash-join.
Refactor: Replace with pullback node PB1 (UP $_{pb}$true).

Payload patch:

{

"structure":{

"nodes":[{"id":"PB1","kind":"pullback","legs":["x: C→K","y: L→K"],"badge":{"UP_pb":true}}],

"edges":[{"src":"C","dst":"PB1","role":"leg1"},{"src":"L","dst":"PB1","role":"leg2"}]

},

"contracts":[{"id":"JoinSpec","over":["C","L"],"kind":"spec","payload":"pairs with same key"}],

"inclusions":[{"id":"InclJoin","graph_of":"PB1","into":"JoinSpec","derivation":{"uses":["UP_pb"],
"sketch":"universal property of pullback"}}]

}

Round-trip test:

- Export DAG; run; log outputs; re-import artifacts with artifact_id unchanged.
- Check SemEq on certified subset; verify UP_pb and InclJoin still hold.

C.3 Executable Witness: CRT Merge over $\mathbb{Z}$

Proof sketch: For coprime m,n, compute s,t with sm+tn=1 (extended GCD).
Witness: merge(u mod m, v mod n) = u·t·n + v·s·m (mod mn).

Witness record:

{

"witnesses":[

```
   {
    "id":"crt_merge",
    "kind":"realizability",
    "ref":"python:arith.crt_merge",
    "hash":"sha256:...",
    "contracts":["CRTSpec"],
    "domain_cert":{"kind":"coprime","payload":{"m": "...", "n":"..."}}
   }
 ]
}
```


Contract spec:


```
{
 "contracts":[
  {"id":"CRTSpec","over":["ResidueM","ResidueN","ResidueMN"],"kind":"spec",
   "payload":"projections recover inputs modulo m and n"}
 ],
 "inclusions":[
  {"id":"InclCRT","graph_of":"crt_merge","into":"CRTSpec","proofref":"sha256:..."}
 ]
}
```


Runtime export: Generates runner.py that checks domain_cert before calling crt_merge; logs include a digest of inputs/outputs for re-verification.

C.4 Sheaf Fusion Mini (two patches)

Setup: Two local predictors p_i: X|_{U_i} → Y|_{U_i} with overlap U_{12}.

Diagram snippet:

```
{
  "structure":{
    "nodes":[
      {"id":"EqOverlap","kind":"equalizer","legs":["p1|U12","p2|U12"],"badge":{"UP_eq":true}},
      {"id":"Glue","kind":"pushout","legs":["incl1: U12→U1","incl2: U12→U2"],"badge":{"UP_po":true}}
    ]
  },
  "contracts":[{"id":"Compat","over":["U12"],"kind":"spec","payload":"p1==p2 on overlap"}],
  "inclusions":[{"id":"InclCompat","graph_of":"EqOverlap","into":"Compat"}],
  "mode_trail":[{"step":0,"mode":"sheaf/local"}]
}
```


Expected: If InclCompat holds, a global section is produced (descent); otherwise the checker reports the earliest failing overlap.

D. CI Hooks (pseudo-commands)

- pl-validate diagram.json → runs type, UP, contract, mode checks; computes artifact_id.
- pl-bridge import --lean Iso1.lean:Group.first_iso → emits patch + artifact_id.
- pl-bridge export --coq diagram.json → emits .v proof obligations.
- pl-runner emit dag diagram.json → writes dag.json, runner.py.
- pl-reverify logs/*.json diagram.json → replays hashes, re-checks inclusions and UPs.
- pl-diff artifact A B → structural diff (objects, arrows, UPs, contracts, modes).

Failure on any check returns non-zero and annotates with one of A.1.1 error codes.

E. Minimal Worked Files (copy/paste)

E.1 Tiny diagram (product + predictor with contract)


```
{
 "version":"1.0",
 "mode":"constructive",
 "objects":[{"id":"F1"},{"id":"F2"},{"id":"F12"},{"id":"P"}],
 "arrows":[
  {"id":"pair","dom":"(F1,F2)","cod":"F12"},
  {"id":"predict","dom":"F12","cod":"P","impl":{"ref":"python:m.predict","hash":"sha256:..."}}
 ],
 "structure":{
  "nodes":[{"id":"Prod","kind":"product","legs":["F1","F2"],"badge":{"UP_times":true}}],
  "edges":[{"src":"F1","dst":"Prod","role":"π1"},{"src":"F2","dst":"Prod","role":"π2"}]
 },
 "contracts":[{"id":"SafeP","over":["P"],"kind":"range","payload":{"min":0,"max":1}}],
 "inclusions":[{"id":"InclSafe","graph_of":"predict","into":"SafeP","proofref":"sha256:..."}],
 "artifact_id":"sha256:TO-BE-COMPUTED"
```

}

E.2 Round-trip record (YAML)

```
roundtrip:
  source:
    tool: lean
    symbol: Group.first_iso
    hash: sha256:AAA
  panlogos:
    artifact: sha256:BBB
    checks: [UP_ok, Contracts_ok, Mode_ok]
  target:
    tool: coq
    file: FirstIso.v
    hash: sha256:CCC
  status: RT-eq, RT-prov
```

F. Implementation Notes

- **Determinism:** normalize diagrams before hashing (sort objects/arrows; alpha-rename automatically).
- **Scalability:** cache UP_* proofs keyed by the shape+legs tuple; invalidate on leg changes only.
- **Safety:** always emit a *mode trail*; block execution if a consumer requires realizability but the artifact lacks a witness.

Closing remark

These APIs, the canonical payload, and the recipes above let teams stand up **PanLogos v1** with measurable guarantees: *import theorems/specs, certify diagrams, extract witnesses, export executable DAGs, and round-trip without semantic drift*.

References

Youvan, D. C. (2025, October). *PanLogos: An Omni-Modal Reflective (∞,1)-Topos Unifying Mathematical Foundations*. ResearchGate. https://www.researchgate.net/publication/396084098_PanLogos_An_Omni-Modal_Reflective_1-Topos_Unifying_Mathematical_Foundations

Awodey, S. (2010). *Category Theory* (2nd ed.). Oxford University Press.

Barr, M. (1970). Exact categories. In *Seminar on Triples and Categorical Homology Theory* (pp. 1–120). Springer LNM 80.

Carboni, A., & Vitale, E. M. (1998). Regular and exact completions. *Journal of Pure and Applied Algebra, 125*(1–3), 79–116.

Cartmell, J. (1986). Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic, 32*(3), 209–243.

Ehresmann, C. (1968). Catégories structurées. *Annales Scientifiques de l'É.N.S.*, 1(3), 1–20.

Giraud, J. (1964). *Méthode de la descente*. Bulletin de la SMF, Mémoire 2.

Goguen, J. A., & Burstall, R. M. (1992). Institutions: Abstract model theory for specification and programming. *Journal of the ACM, 39*(1), 95–146.

Hyland, J. M. E. (1982). The effective topos. In A. S. Troelstra & D. van Dalen (Eds.), *The L. E. J. Brouwer Centenary Symposium* (pp. 165–216). North-Holland.

Johnstone, P. T. (2002). *Sketches of an Elephant: A Topos Theory Compendium* (Vols. 1–2). Oxford University Press.

Joyal, A., & Moerdijk, I. (1995). *Algebraic Set Theory*. Cambridge University Press.

Joyal, A., & Street, R. (1991). The geometry of tensor calculus I. *Advances in Mathematics, 88*(1), 55–112.

Kelly, G. M. (1982). *Basic Concepts of Enriched Category Theory*. Cambridge University Press.

Lawvere, F. W. (1963). Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences, 50*(5), 869–872.

Lawvere, F. W. (1964). An elementary theory of the category of sets (ETCS). *Proceedings of the National Academy of Sciences, 52*(6), 1506–1511.

Lawvere, F. W., & Tierney, M. (1970). Quantifiers and sheaves. In *Actes du Congrès International des Mathématiciens* (Nice, 1970), Tome 1 (pp. 329–334). Gauthier-Villars.

Lurie, J. (2009). *Higher Topos Theory*. Princeton University Press.

Lurie, J. (2017). *Higher Algebra* (book draft).

Mac Lane, S. (1998). *Categories for the Working Mathematician* (2nd ed.). Springer.

Makkai, M., & Paré, R. (1989). *Accessible Categories: The Foundations of Categorical Model Theory*. AMS.

Meseguer, J. (1992). Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science, 96*(1), 73–155.

Riehl, E. (2016). *Category Theory in Context*. Dover.

Selinger, P. (2010). A survey of graphical languages for monoidal categories. In *New Structures for Physics* (pp. 289–355). Springer.

Shulman, M. (2008). Exact completions and small sheaves. *Theory and Applications of Categories, 22*(18), 436–491.

Street, R. (1972). The formal theory of monads. *Journal of Pure and Applied Algebra, 2*(2), 149–168.

The Univalent Foundations Program. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. IAS.

Tierney, M. (1972). Sheaf theory and the continuum hypothesis. In *Toposes, Algebraic Geometry and Logic* (pp. 13–42). Springer LNM 274.

van Oosten, J. (2008). *Realizability: An Introduction to its Categorical Side*. Elsevier.

Wraith, G. C. (1972). Artin glueing. *Journal of the London Mathematical Society, 2*(2), 319–326.

Wyler, O. (1991). *Algebraic Theories and Categorical Structures*. World Scientific.

Optional AI-focused complements (for this paper's applications):

Bauckhage, C., & Kersting, K. (2013). Data mining and pattern recognition in sheaves and simplicial complexes. *KI*, 27(4), 345–351.

Davies, E., et al. (2021). Advancing mathematics by guiding human intuition with AI. *Nature, 600*, 70–74.

Fong, B., & Spivak, D. I. (2019). *Seven Sketches in Compositionality: An Invitation to Applied Category Theory*. Cambridge University Press.

Garlan, D., & Schmerl, B. (2002). Model-based adaptation for self-healing systems. In *WOSS* (pp. 27–32).

Ghica, D. R., & Smith, A. (2020). Compositional game semantics for program synthesis. *LMCS, 16*(4), 1–43.

Sadrzadeh, M., Coecke, B., & Clark, S. (2013). The categorical compositional distributional model of meaning. *Linguistics and Philosophy, 36*(1), 1–31.

Spivak, D. I., & Tan, Z. (2019). Sheaves as semantic data structures. *arXiv:1810.06067*.