ai1-Linguo: A CT/HoTT Compliance Bench for Language on a Single PC

Douglas C. Youvan

doug@youvan.com

www.youvan.ai

September 15, 2025

Building on *Achieving Artificial Intelligence (youvan.ai1) on a PC with Category Theory & HoTT*, we present ai1-Linguo, a compact, data-driven bench that turns claims about CT/HoTT semantics into executable compliance tests. Instead of optimizing losses, the bench audits whether linguistic operations respect algebraic laws and homotopical structure. A CSV interface feeds typed Yoneda probes for meaning-as-use (reporting *YonedaSynonym*, *YonedaNonSyn*, and YonedaAUC), pullback coreference (*PCS*), a normalized Beck–Chevalley scope score, sheaf gluing for discourse coherence, transport invariance under licensed substitutions, and a Paraphrase Homotopy Index (*PHI* and node-normalized *PHI_nodes*) that detects higher-path cycles in paraphrase graphs. We add three structural checks: a univalence witness for concept equivalence, curry/uncurry parity for functions as adjoints, and a presupposition monad with an effect-leak audit. Every run completes on a single Windows PC, emits human-readable JSON, and auto-generates an HTML summary plus Graphviz DOT for visualization—supporting replication, pedagogy, and forensic debugging. We position ai1-Linguo as a complementary path to modern AI: a local-first, deterministic pipeline where explanations are primary artifacts and scores are by-products of verified equations. The result is a small, inspectable system that makes CT/HoTT legible to practitioners while remaining strong enough to falsify modeling choices.

Keywords: ai1, category theory, homotopy type theory, HoTT, compositional semantics, compliance bench, Yoneda lemma, Yoneda probes, YonedaAUC, pullback coreference, PCS, Beck–Chevalley, scope, sheaf gluing, transport invariance, paraphrase graph, Paraphrase Homotopy Index, PHI_nodes, univalence witness, curry–uncurry parity, presupposition monad, effect-leak audit, deterministic rewriting, equational reasoning, CSV datasets, JSON traces, HTML report, Graphviz DOT, local-first AI, single-PC reproducibility, interpretability, Windows PowerShell

A collaboration with GPT-5. CC4.0. 82 pages.

Outline

1. **Introduction & Contributions**
    - Motivation: executable semantics vs. loss minimization
    - Problem statement and scope
    - Key contributions (bench, metrics, artifacts, reproducibility)
2. **Background & Positioning**
    - CT/HoTT notions used (objects, morphisms, pullbacks, univalence, curry–uncurry)
    - Relation to compositional semantics, type-theoretic pragmatics, and symbolic AI
    - How this complements LLMs and statistical NLP
3. **System Overview (ai1-Linguo Architecture)**
    - Pipeline: CSV → metrics → JSON → HTML/DOT
    - Determinism, local-first execution, and audit trail
    - Extensibility points (new metrics, loaders, exporters)
4. **Formal Core (CT/HoTT Framing)**
    - Categories of expressions and functoriality of composition
    - Pullbacks as constraint solving; sheaves and gluing for discourse
    - Beck–Chevalley as scope-commutation; homotopies for paraphrase cycles
    - Univalence as concept equivalence policy; adjunction for curry–uncurry
    - Effects and monads for presuppositions
5. **Metrics Suite (Compliance Battery)**
   5.1 Functor Consistency (associativity/identity)
   5.2 Yoneda Probes: Synonym/Non-Syn scores + **YonedaAUC**
   5.3 Pullback Coreference Score (PCS)
   5.4 Beck–Chevalley Defect (normalized)
   5.5 Sheaf Gluing Ratio (constraint compatibility)
   5.6 Transport Invariance under licensed substitutions

5.7 Paraphrase Homotopy Index (**PHI** and **PHI_nodes**)
5.8 Univalence Witness (round-trip equivalences)
5.9 Curry/Uncurry Parity (Hom(X×A,B) ≅ Hom(X,B^A))
5.10 Presupposition Monad & Effect-Leak Audit

6. **Data Interface & Schemas**
    - CSV specs: probe_truth.csv, synonyms.csv, mentions.csv, pronouns.csv, paraphrases.csv, discourse.csv, scope.csv, transport.csv
    - Coverage requirements and the probe_sync.py utility
    - Provenance and licensing notes
7. **Implementation Details**
    - Files and responsibilities (category.py, hott.py, metrics.py, cthott_bench.py, cthott_export.py)
    - Complexity and runtime on a single PC
    - Exports: JSON report, HTML report, Graphviz DOT/PNG
    - Automation (run_all.bat) and PowerShell workflow
8. **Results: Sample Runs & Sensitivity**
    - Baseline scores on included toy data
    - Sensitivity to added probes/pairs/edges
    - Visualizing paraphrase geometry (DOT/PNG)
    - Interpreting **PHI** vs **PHI_nodes**
9. **Case Studies**
    - Synonym drift under probe changes
    - Coreference disambiguation via pullbacks
    - Scope illusions and BC-defect shifts
    - Discourse (un)glueability in conflicting timelines

10. **Falsification Harness & Thresholds**
    - Turning metrics into gates/alarms
    - Example failure modes and how they surface
11. **Validation & Calibration**
    - Human judgments vs. YonedaAUC alignment (procedure)
    - Bootstrap/perturbation tests for metric stability
12. **Reproducibility & Auditability**
    - Deterministic runs, seed policy (if any), and JSON artifacts
    - Diff-based change tracking across versions
13. **Comparison to Learning-Centric Systems**
    - Where ai1-Linguo fits; hybrid flows (LLM as data generator; bench as verifier)
14. **Limitations & Threats to Validity**
    - Toy scale; probe subjectivity; dataset bias
    - Over-interpretation risks; scope of CT/HoTT idealizations
15. **Ethical Considerations**
    - Transparency, interpretability, and responsible evaluation
    - Avoiding misuse of “compliance” scores
16. **Future Work**
    - Larger probe banks; richer sheaf models; higher-category moves
    - Integration with ai1 term-rewriting engine and GUI
    - Community datasets and plug-in metrics
17. **Conclusion**
    - Summary of what executable CT/HoTT brings to language evaluation
    - Invitation to reproduce and extend

1. Introduction & Contributions

Motivation: executable semantics vs. loss minimization

Mainstream language systems are trained to **minimize loss**, not to **obey laws**. Cross-entropy and RLHF yield powerful correlational behavior, yet they do not guarantee that basic semantic invariants—associativity of composition, lawful scope movement, transport under paraphrase—are respected. When these invariants fail, models can be brittle, hard to audit, and difficult to debug. Our premise is simple: for many linguistic phenomena the right question is not "did the model predict token *t*?", but "did the computation preserve the **equation** it should preserve?" We therefore propose **executable semantics**: a small, local bench that evaluates whether language transformations satisfy **Category Theory (CT)** and **Homotopy Type Theory (HoTT)** laws, with machine-checkable witnesses and human-readable artifacts.

Problem statement and scope

There is no lightweight, reproducible toolchain that (a) instantiates CT/HoTT notions on concrete language data, (b) scores models or rule systems against those notions, and (c) runs deterministically on a single PC with verifiable outputs. We target that gap. The present workbench—**ai1-Linguo**—focuses on: (i) compositional semantics (functoriality, pullbacks, adjunctions), (ii) scope and discourse structure (Beck–Chevalley, sheaf gluing), (iii) paraphrase geometry (homotopies, cycles), and (iv) effectful pragmatics (presuppositions). Scope is deliberately **toy-to-small** but **extensible**: datasets are CSVs; exports are JSON/HTML/DOT; everything runs offline on Windows/PowerShell (Graphviz optional).

Key contributions

1. **Bench architecture.** A deterministic pipeline: CSV → metrics → JSON → HTML/DOT, plus a one-click runner and coverage checker.
2. **Metrics suite (compliance battery).**
    - **FunctorConsistency** (assoc/identity), **YonedaSynonym / YonedaNonSyn / YonedaAUC** (meaning-as-use probes), **PCS** (pullback coreference), **BCDefectNorm** (scope commutation), **Gluability** (sheaf compatibility), **Transport** (licensed substitution), **PHI** and **PHI_nodes** (paraphrase cycles), **UA_witness** (univalence round-trip), **CurryParity** (Hom(X×A,B) ≅ Hom(X,B^A)), **EffectLeakAudit** (presupposition monad).
3. **Executable artifacts.** Human-readable **JSON** reports, **HTML** summaries, **Graphviz DOT/PNG** graphs, and reproducible logs; all produced locally.
4. **Reproducibility & falsification.** No gradients, no randomness—runs are auditable; thresholds can be used as **gates** to accept/reject modeling choices.
5. **Bridging CT/HoTT and practice.** The bench operationalizes abstract laws on real text pairs and small graphs, making the mathematics legible and testable on a single PC.

2. Background & Positioning

CT/HoTT notions used (and how we instantiate them)

- **Objects & morphisms.**
  We model small "types of expressions" as **objects** (e.g., N for noun phrases, S for sentences) and meaning-preserving transformations as **morphisms** (e.g., very : N→N, say : N→S). The bench checks the **category laws**:
    - **Identity:** id_X ∘ f = f = f ∘ id_Y.
    - **Associativity:** h ∘ (g ∘ f) = (h ∘ g) ∘ f.
      In code these are computed on concrete strings so that lawfulness is *executable*, not assumed.
- **Functors & naturality (intuitively).**
  A functor maps objects/morphisms in "syntax" to objects/morphisms in "evaluation," preserving identities and composition. Our active↔passive normalization behaves like a functor: if substitution after paraphrase equals paraphrase after substitution, **transport** holds.

- **Pullbacks.**
  A pullback is a limit that enforces **feature agreement** by solving equations in a square. We use pullbacks for **coreference**: an anaphor and a candidate antecedent must match gender/number (and optionally discourse role/recency). The **PCS** score reports how often the pullback picks the gold antecedent.

- **Beck–Chevalley (BC).**
  BC is a **commutation law** for substitution and quantification. Where cartesian structure licenses a scope move, the two paths around the square should agree. We compute a **BC-defect** from paired scores to quantify how much a system prefers the lawful path.

- **Sheaves & gluing.**
  Local constraints over overlapping "covers" should agree to form a global discourse interpretation. The **Gluability** ratio counts families that can be glued (no conflicts on overlaps).

- **Homotopies & paraphrase cycles.**
  Viewing paraphrases as edges yields **higher-path** structure: returning to a sentence along distinct paraphrase routes forms a loop. We define **PHI** (fraction of nontrivial cycles) and **PHI_nodes** (nontrivial cycles per node) to detect when paraphrase is more than pairwise equivalence.

- **Univalence (witnessed equivalence).**
  Two "concept types" may be treated as equal when there is a **bidirectional equivalence with round-trip witnesses**. Our UA_witness only collapses a pair if both directions compose to the identity up to normalization.

- **Curry–uncurry (adjunction).**
  The isomorphism Hom(X×A, B) ≅ Hom(X, B^A) grounds the intuition that predicates with arguments are equivalent to functions returning functions. We check parity on a tiny world (students×papers).

- **Monads for effects.**
  Presuppositions are modeled as effects that thread through composition. The **EffectLeakAudit** ensures removing a trigger doesn't silently flip main-clause truth, surfacing uncontrolled side effects.

Relation to compositional semantics, type-theoretic pragmatics, and symbolic AI

- **Compositional semantics.**
  The tradition from Montague forward builds sentence meanings compositionally. Category-theoretic treatments make the **laws** explicit (identities, associativity, functorial

mappings), and our bench turns those laws into tests. Where distributional or neural models give similarity, we additionally ask: *does the transformation respect the equation?*

- **Type-theoretic pragmatics.**
  Dependent types and algebraic effects have long modeled context, presupposition, and anaphora. We adopt a **minimal executable fragment**: pullbacks for agreement, a presupposition **monad** for effect tracking, and sheaf-style gluing for discourse compatibility. The emphasis is not to outdo full formalisms, but to **operationalize** them so a practitioner can run, visualize, and falsify.

- **Symbolic AI & rewriting.**
  Classic symbolic systems provided **explanations** but often lacked scalable evaluation harnesses. Our design borrows from equational reasoning and term rewriting (laws, normal forms, witnesses) and adds a **compliance battery** that yields numeric summaries (JSON/HTML) and structural artifacts (DOT graphs), giving symbolic methods a modern, auditable workflow.

How this complements LLMs and statistical NLP

- **Division of labor.**
  LLMs are superb at generating **candidates** (paraphrases, coref features, probe labels). The bench is superb at **verifying structure**: it tells you which candidates respect CT/HoTT laws. Together you get *breadth* from learning and *soundness signals* from equations.

- **Constraint-guided data curation.**
  Use an LLM to propose paraphrase edges; keep only those that improve **PHI_nodes** or reduce **BC-defect**. Over time, you curate datasets that exhibit the right algebraic geometry.

- **Regularization & training signals.**
  Compliance metrics can become **auxiliary losses** or selection criteria: e.g., reward models that preserve **transport invariance** or that maintain high **YonedaAUC** while lowering **YonedaNonSyn**.

- **Auditing & regression tests.**
  When a new model version ships, run the local bench. If **Gluability** drops or **PCS** collapses on your datasets, you catch regressions early—no cloud infrastructure required.

- **Interpretability bridge.**
  The bench outputs **machine-checkable witnesses** and human-readable reports. This

gives teams a way to discuss behavior in terms of laws and counterexamples, not just aggregate accuracy.

In short, ai1-Linguo operationalizes CT/HoTT primitives in a form that is *complementary* to data-driven NLP: the model proposes; the mathematics disposes.

3. System Overview (ai1-Linguo Architecture)

Pipeline: CSV → metrics → JSON → HTML/DOT

Layers & files

- **Data (CSV):** data/*.csv — tiny, human-editable datasets (probe_truth.csv, synonyms.csv, mentions.csv, pronouns.csv, paraphrases.csv, discourse.csv, scope.csv, transport.csv).
- **Core library:**
  - category.py — objects/morphisms, composition, pullback.
  - hott.py — paraphrase graph, cycle finder, PHI/PHI_nodes, transport-invariance.
  - metrics.py — Probe class, Yoneda similarity, PCS, gluing, BC-defect helpers.
- **Orchestrator:** cthott_bench.py — loads CSV → calls metrics → assembles **Compliance Report** dict.
- **Exporter:** cthott_export.py — reads JSON → writes **bench_report.html**, **paraphrase_graph.dot**, and **probe_matrix.csv**.
- **Utilities:** probe_sync.py (fills missing probe values), run_all.bat (one-click bench→export→PNG→open).

Execution flow

1. **Load CSVs** (ensure_sample_data() creates examples if empty).
2. **Compute metrics** (all return floats in 0,10,10,1):
   FunctorConsistency, YonedaSynonym, YonedaNonSyn, YonedaAUC, PCS, BCDefectNorm, Gluability, Transport, PHI, PHI_nodes, UA_witness, CurryParity, EffectLeakAudit.
3. **Write JSON** → out/bench_report.json (single source of truth).
4. **Export** → out/bench_report.html (readable table), out/paraphrase_graph.dot (Graphviz), out/probe_matrix.csv (pivot of probe truths).

5. *(Optional)* Render PNG with Graphviz: dot -Tpng out\paraphrase_graph.dot -o out\paraphrase_graph.png.

Determinism, local-first execution, and audit trail

- **Deterministic by construction:** no randomness, no gradients, no network calls. Cycle enumeration, node orders, and tables are **sorted** so identical CSVs produce identical scores and DOT structure.
- **Local-first:** runs fully on a single Windows PC (PowerShell). Dependencies are standard Python; Graphviz is optional for PNG rendering.
- **Audit artifacts:**
    - **out/bench_report.json** — machine-readable record of every score.
    - **out/bench_report.html** — human summary with timestamps.
    - **out/paraphrase_graph.dot / .png** — structural evidence for PHI/PHI_nodes.
    - **out/probe_matrix.csv** — explicit truth table used for Yoneda scores.
      Put data/ and out/ under version control to track changes; diff JSON between commits for regression checks.

Extensibility points (new metrics, loaders, exporters)

A. New metrics
Add a pure function in cthott_bench.py (or a helper in metrics.py) that returns a float in 0,10,10,1. Typical pattern:

```
def my_metric_from_csv() -> float:
    rows = read_csv(os.path.join(DATA, "myfile.csv"))
    # compute score in [0,1]
    return score
```

Then:

1. Call it in main() and insert into the scores dict.
2. Add its key to the report table order.

3. If it needs new primitives, extend category.py/hott.py minimally and keep functions **side-effect free**.

B. New loaders (CSV schemas)
Create data/myfile.csv with a small header; implement read_csv() use as above. For coverage-style checks, mirror probe_sync.py to ensure completeness.

C. New exporters
Clone the pattern in cthott_export.py: read out/bench_report.json and render to your target (Markdown, LaTeX, or a REST payload). Because JSON is the truth, exporters are independent and can be swapped in/out without touching metrics.

D. Swap-in data generators (LLM, rules, or scripts)
Upstream any generator that writes the same CSV schemas. The bench doesn't care where CSVs came from—only that they're well-formed—so you can test alternative pipelines against the **same** compliance gates.

Design invariants for extensions

- **Purity:** metric = pure function of CSVs (no I/O outside data/).
- **Monotone surfaces:** prefer scores with clear interpretation (0=fail, 1=pass).
- **Explainability:** whenever possible, emit auxiliary artifacts (e.g., a DOT, a small CSV of counterexamples) to accompany each new score.
- **Determinism:** avoid nondeterministic iteration or hashing; sort keys before traversals.

This architecture keeps the bench small, inspectable, and easy to grow—new phenomena become new CSVs + one function + one row in the report.

4. Formal Core (CT/HoTT Framing)

Categories of expressions and functoriality of composition

Objects are small expression types (e.g., N, S, 1). **Morphisms** are meaning-preserving maps (e.g., very:N→N, say:N→S). We require the category laws:

- **Identities:** id_X ∘ f = f = f ∘ id_Y
- **Associativity:** h ∘ (g ∘ f) = (h ∘ g) ∘ f

In the bench, we interpret morphisms as total Python functions and *execute* these equalities on concrete payloads (strings). A lightweight "evaluation" functor carries syntax to values; our

transport-invariance check is a naturality-style sanity test: meaning doesn't change if you apply a licensed paraphrase before/after a downstream evaluator.

Why it matters. If composition and identities don't hold operationally, every higher construction (pullbacks, adjunctions) becomes unreliable. Functoriality makes "do it then evaluate" = "evaluate then do it."

Pullbacks as constraint solving; sheaves and gluing for discourse

A **pullback** of A→C←B is the object A×_C B together with projections making the square commute and universal. Linguistically, think "meet all features that must agree." For coreference:

- f: Candidate → (gender,number,…)
- g: Pronoun → (gender,number,…)
- Pullback pairs (a,p) satisfy f(a)=g(p).

We compute a **PCS** score by selecting the candidate(s) in the pullback and comparing to gold; tie-breakers (e.g., subject preference, recency) are external but transparent.

A **presheaf** assigns each discourse region its local constraints; **gluing** asks whether locally consistent pieces agree on overlaps to form a global reading. Our **Gluability** metric is the fraction of families that admit such a glue (no explicit conflicts on shared keys).

Why it matters. Coreference and discourse coherence are solved as *limits*: you don't memorize answers—you satisfy equations.

Beck–Chevalley as scope-commutation; homotopies for paraphrase cycles

The **Beck–Chevalley** (BC) condition says certain squares commute when you exchange substitution with quantification/aggregation. In scope phenomena, the "lawful" (cartesian-licensed) path and the "illicit" path should agree; violations show up as a **BC-defect**. We expose this by pairing minimal items (e.g., ∀/∃ reorderings) and scoring (cartesian − noncartesian); larger is more lawful.

A **paraphrase graph** has sentences as nodes and paraphrase links as edges. Distinct paths from s back to s form **loops**—syntactic/semantic **homotopies**. Nontrivial cycles (length > 2) indicate higher-path structure (not mere symmetric equivalence). We report:

- **PHI:** nontrivial cycles / all cycles (legacy, scale-sensitive)
- **PHI_nodes:** nontrivial cycles / #nodes (scale-free "cycles per node")

Why it matters. BC captures lawful movement; homotopies reveal the *geometry* of paraphrase beyond pairwise matching.

Univalence as concept equivalence policy; adjunction for curry–uncurry

Univalence (HoTT intuition). Types are equal iff they are **equivalent**. Operationally, we require round-trip witnesses:

$A \xrightarrow{e} B$, $B \xrightarrow{e^{-1}} A$

$e^{-1} \circ e \sim id_A$ and $e \circ e^{-1} \sim id_B$ (up to normalization)

Only then may we "collapse" the concepts. The bench's UA_witness checks these equalities on finite samples (e.g., *bachelor ⇄ unmarried adult male*).

Adjunction / Curry–uncurry. The set-level isomorphism

$\mathrm{Hom}(X \times A, B) \cong \mathrm{Hom}(X, B^A)$

is tested by enumerating X×A and verifying parity between the uncurried predicate and its curried form. Linguistically, this validates treating a binary predicate as a function returning a predicate.

Why it matters. Univalence guards against premature concept identification; curry–uncurry guarantees that "adding an argument" is structurally the same as "producing a function."

Effects and monads for presuppositions

Presuppositions behave like **effects** that propagate through composition. We use a tiny **Presupposition monad**:

- Carrier: M P = (truth: Bool, effects: Set[Presupp])
- **unit:** $x \mapsto (x, \emptyset)$
- **bind:** $(v,E) >>= k = \text{let } (v',E') = k(v) \text{ in } (v', E \cup E')$

Triggers (e.g., *stop*) contribute effects (john_smoked_before). The **Effect-Leak Audit** compares main-clause truth with vs. without the trigger; differences indicate uncontrolled effect leakage.

In well-disciplined pipelines, truth remains stable while effects accumulate for later checking/repair.

Why it matters. This separates *what is asserted* from *what must already hold*, making side effects explicit and auditable—exactly the kind of separation CT/HoTT encourages.

Takeaway. Each metric in the bench is a compiled instance of these constructions: identities/associativity (category), pullbacks (limits), BC (commutation), homotopies (higher paths), univalence (equivalences), adjunction (curry–uncurry), and monads (effects). Because they are *executed* on concrete data, the abstractions become falsifiable engineering checks rather than purely theoretical aspirations.

5. Metrics Suite (Compliance Battery)

Below each metric you'll find: **Inputs** (CSV fields), **Computation** (what we check), **Output & interpretation** (0..1), **Failure modes**, and **Extensibility**.

5.1 Functor Consistency (associativity/identity)

Inputs. None (built-in smoke test).

Computation. Treat small expression types as objects (1, N, S) and string transformers as morphisms (building:1→N, tall:N→N, very:N→N, say:N→S). Execute:

- Identity: id_S ∘ (say ∘ very ∘ tall ∘ building) = say ∘ very ∘ tall ∘ building
- Associativity: h ∘ (g ∘ f) = (h ∘ g) ∘ f on the same payload.

Output & interpretation. 1.0 if both equalities hold, else 0.0. This guards the entire stack.

Failure modes. Non-pure morphisms; hidden state; order-dependent normalization.

Extensibility. Add more chains and payloads; score is the mean over tests.

5.2 Yoneda Probes: Synonym/Non-Syn + YonedaAUC

Inputs.

- probe_truth.csv: (probe,word,value∈{0,1})
- synonyms.csv: (w1,w2,label∈{1,0})

Computation. For each pair (w1,w2), similarity sim = mean_p [1{ value_p(w1)=value_p(w2) }]. Report:

- **YonedaSynonym** = mean(sim | label=1)
- **YonedaNonSyn** = 1 − mean(sim | label=0)
- **YonedaAUC** = Mann–Whitney probability that a random positive outscores a random negative (ties = 0.5).

Output & interpretation. Higher is better. AUC near 1.0 means probe patterns align with your synonym labels.

Failure modes. Missing probe cells; overly generic probes (everything matches); contradictory probe sets.

Extensibility. Add weights per probe; graded values (e.g., {0,½,1}); export per-pair scores for diagnostics.

5.3 Pullback Coreference Score (PCS)

Inputs.

- mentions.csv: (id,gender,num,role,recency,…)
- pronouns.csv: (form,gender,num,gold)

Computation. Pullback over feature maps: f(cand) = (gender,num,…), g(pron)=(gender,num,…). Candidates that satisfy f= g form the fiber product. Rank (subject≺object; higher recency first). Chosen id is compared to gold.

Output & interpretation. PCS = accuracy over pronouns (0..1). Clear, falsifiable coref primitive.

Failure modes. Sparse or wrong features; tie-breaking dominates (tuneable).

Extensibility. Add features (person, animacy, case); plug in learned ranking but keep the pullback filter.

5.4 Beck–Chevalley Defect (normalized)

Inputs.

- scope.csv: rows with cond∈{cartesian, noncartesian}, score∈[0,1] from a model or heuristic.

Computation. Let C = mean(score | cartesian), N = mean(score | noncartesian).
BCDefectNorm = clamp[0,1]_{[0,1]}[0,1]((C − N + 1)/2).

Output & interpretation. Higher means the lawful (commuting) path is preferred.

Failure modes. Unbalanced stimuli; scores not comparable across items.

Extensibility. Separate by construction type (quantifier raising vs. trace movement); report per-item defects.

5.5 Sheaf Gluing Ratio (constraint compatibility)

Inputs.

- discourse.csv: (family,key,value). Mark conflicts as tense="conflict: …".

Computation. Group by family. A family is **gluable** if no key has incompatible values on overlaps.
Gluability = (# gluable families) / (total families).

Output & interpretation. Fraction of local readings that form a global reading.

Failure modes. Over-permissive unification (everything glues); or brittle equality (nothing glues).

Extensibility. Replace string equality with typed unification (e.g., intervals for time, lattices for topics).

5.6 Transport Invariance (licensed substitutions)

Inputs.

- transport.csv: (s1,s2) sentence pairs licensed as “same meaning”.

Computation. Provide an evaluator E(s) (here: canonical bag-of-content words).
Transport = meanpairs_\text{pairs}pairs[1{ E(s1)=E(s2) }].

Output & interpretation. 1.0 means your evaluator is indifferent to the licensed transformation (e.g., active↔passive).

Failure modes. Over-aggressive normalization that collapses genuinely different pairs; evaluator too weak.

Extensibility. Swap E for a typed semantic form or a rule-based DRS; report mismatch diffs.

5.7 Paraphrase Homotopy Index (PHI, PHI_nodes)

Inputs.

- paraphrases.csv: (s1,s2,reason) edges in a paraphrase graph.

Computation. Find simple cycles; mark **nontrivial** if length > 2.

- **PHI** = (# nontrivial cycles) / (all cycles) — legacy, scale-sensitive.
- **PHI_nodes** = (# nontrivial cycles) / (# nodes) — scale-free "cycles per node".

Output & interpretation. Higher values indicate richer higher-path structure (paraphrase beyond symmetric pairs).

Failure modes. Duplicate edges, near-duplicates, or trivial 2-cycles dominating.

Extensibility. Weight edges by "reason"; compute homotopy classes modulo neutral moves; export counterexample loops.

5.8 Univalence Witness (round-trip equivalences)

Inputs. Built-in samples (can be generalized to CSV).

Computation. Provide e: A→B, e^{-1}: B→A and a normalizer ν.
Check $\nu(e^{-1} \circ e(a)) = \nu(a)$ for sample $a \in A$, and $e \circ e^{-1}(b) = b$ for $b \in B$.
UA_witness = 1.0 if both hold on the set; else 0.0.

Output & interpretation. Guards concept collapse: identification only under witnessed equivalence.

Failure modes. Over-strong or under-strong normalization; biased samples.

Extensibility. Load multiple (A,B,e,e^{-1}) specs from CSV; report per-pair pass rates.

5.9 Curry/Uncurry Parity (Hom(X×A,B) ≅ Hom(X,B^A))

Inputs. Built-in finite sets X, A and relation Read ⊆ X×A.

Computation. Compare uncurried f:X×A→B with curried g:X→(A→B) on all pairs.
CurryParity = mean(x,a)_{(x,a)}(x,a)[1{ f(x,a)=g(x)(a) }].

Output & interpretation. 1.0 shows the expected adjoint shape holds empirically in your mini-world.

Failure modes. Impure functions; inconsistent closures.

Extensibility. Drive X,A,Read from CSV; test higher-arity curryings.

5.10 Presupposition Monad & Effect-Leak Audit

Inputs. Built-in triggers; (extendable to CSV of (trigger, main-clause) pairs).

Computation. Model (truth, effects) with **unit** and **bind**; a trigger like *stopped* adds john_smoked_before. Compare main-clause truth **with** trigger vs. **without** trigger.
EffectLeakAudit = 1.0 if truths match (effects isolated), else 0.0.

Output & interpretation. Detects uncontrolled entanglement between asserted content and presupposed context.

Failure modes. Treating effects as truth; missing effect propagation at composition points.

Extensibility. Batch over CSV; report (added, required, unhandled) effect sets per item; score as 1 − leak_rate.

Practical notes (all metrics)

- **Range.** All scores are normalized to [0,1].
- **Artifacts.** Every run emits out/bench_report.json (scores), out/bench_report.html (table), out/paraphrase_graph.dot/png (structure), and out/probe_matrix.csv (Yoneda surface).
- **Thresholding.** Treat scores as **gates** (e.g., YonedaAUC ≥ 0.85, Transport ≥ 0.95, PHI_nodes rising with more curated edges).
- **Diagnostics.** When a score drops, inspect the corresponding CSV and exported artifacts (e.g., loops in DOT, mismatching probe rows) to locate the cause.

6. Data Interface & Schemas

Overview

ai1-Linguo is **data-first**: every metric reads a tiny CSV with a fixed header and permissive extras (unused columns are ignored). Files are UTF-8, comma-separated, one header row, quotes required when fields contain commas or quotes. All IDs/strings are case-sensitive unless noted.

CSV specs

6.1 probe_truth.csv

Maps **probes** (typed "uses/contexts") to binary truth on words.

Header

probe,word,value

- probe (str): e.g., finance?, countable?.
- word (str): token or lemma.
- value (0|1): integer.

Example

finance?,bank,1

countable?,river,1

Notes

- Extra columns are ignored (you may add source, annotator, …).
- Used by: **YonedaSynonym / YonedaNonSyn / YonedaAUC**.

6.2 synonyms.csv

Pairs to evaluate under probes.

Header

w1,w2,label

- w1,w2 (str)
- label (1|0): 1 = synonym / near-synonym; 0 = non-synonym.

Example

rock,anvil,1

bank,river,0

Notes

- Coverage expectation: every word in this file should appear in probe_truth.csv for **all** probes (see §6.9).

6.3 mentions.csv

Candidate antecedents with features for pullback coreference.

Header

id,gender,num,role,recency

- id (str): unique mention label (e.g., Alice).
- gender (f|m|n|x|…): your scheme; matcher uses equality.
- num (sg|pl|…): your scheme; matcher uses equality.
- role (subj|obj|…): used only for tie-break ranking.
- recency (int ≥0): higher = more recent (used for tie-break).

Example

Alice,f,sg,subj,2

Beth,f,sg,obj,1

6.4 pronouns.csv

Pronouns (anaphors) with gold antecedents.

Header

form,gender,num,gold

- form (str): pronoun surface (e.g., she).
- gender,num (as above).

- gold (str): gold antecedent id.

Example

she,f,sg,Alice

Notes

- Used by: **PCS (pullback coreference accuracy)**.

6.5 paraphrases.csv

Directed paraphrase edges; exporter writes an undirected look by duplicating edges.

Header

s1,s2,reason

- s1,s2 (str): sentences; **must be quoted** if commas present.
- reason (str, optional): label such as active-passive, nominalization.

Example

"The committee approved the plan.","The plan was approved by the committee.","active-passive"

Notes

- Used by: **PHI**, **PHI_nodes**; exporter emits out/paraphrase_graph.dot.

6.6 discourse.csv

Local constraints per “family” (mini discourse). Glues if no key conflicts.

Header

family,key,value

- family (str/int): grouping id.
- key (str): e.g., topic, tense, agent.
- value (str): literal or special **conflict** marker.

Conflict convention

- Any value whose lowercase starts with conflict: is treated as an **explicit clash** for that key.

Example

1,topic,budget

1,tense,past

2,tense,conflict: past vs. present

Notes

- Used by: **Gluability** (gluing ratio).

6.7 scope.csv

Paired scores for scope items under lawful vs. illicit moves.

Header

cond,score

- cond (cartesian|noncartesian)
- score (float in [0,1])

Example

cartesian,0.95

noncartesian,0.25

Notes

- Used by: **BCDefectNorm** = f(mean(cartesian) − mean(noncartesian)).

6.8 transport.csv

Pairs licensed to be “same meaning” under allowed transformations (e.g., active↔passive).

Header

s1,s2

Example

"The bank approved the loan.","The loan was approved by the bank."

Notes

- Evaluated via a simple canonicalizer in the repo; you can swap in your own evaluator.

6.9 Coverage requirements and probe_sync.py

Why coverage matters

Yoneda metrics compare probe **patterns** across words. If a word in synonyms.csv is missing a value for a probe in probe_truth.csv, the bench can still run (defaults to 0 in the raw-score path) but your signal degrades and **coverage warnings** appear.

Two guardrails

1. **Yoneda coverage warnings** (already in cthott_bench.py) list (w1,w2,probe) triples with missing cells (prints top 10).
2. **probe_sync.py** (you installed it): fills **all missing (probe,word)** cells with a default value (--fill 0 or --fill 1) and rewrites probe_truth.csv in sorted order.

Typical use

python .\probe_sync.py --fill 0

python .\cthott_bench.py

Best practice

- Run probe_sync.py whenever you add **new words** to synonyms.csv or **new probes** to probe_truth.csv.
- After syncing, open out\probe_matrix.csv (exporter) to eyeball the matrix.

6.10 Provenance and licensing notes

Provenance fields (recommended)

Although loaders ignore extra columns, we **recommend** adding light metadata columns to each CSV to enable auditing:

- source (str/url): where the item came from (curated, synthetic, link).

- annotator (str/id): who labeled it.
- date (YYYY-MM-DD): annotation date.
- license (str): short code (CC-BY-4.0, CC0, ODC-By-1.0, Proprietary-Internal).
- notes (str): freeform.

Example (probe_truth.csv)

probe,word,value,source,annotator,date,license

finance?,bank,1,curated,dyouvan,2025-09-14,CC0

Suggested dataset manifest

Add a top-level data/manifest.json to capture dataset-level meta:

```
{
  "name": "ai1-linguo-toyset",
  "version": "0.3.0",
  "description": "Tiny CSVs for CT/HoTT compliance bench.",
  "created": "2025-09-14",
  "license": "CC-BY-4.0",
  "provenance": ["hand-curated", "synthetic paraphrases"],
  "contact": "youvan.ai1@example.org"
}
```

Reproducibility etiquette

- Commit data/ and out/bench_report.json to version control.
- Add a simple CHANGELOG.md noting dataset changes (probes added, pairs added/removed).
- Tag releases (e.g., v0.3.0) so scores are comparable across time.

Editing & validation tips (quick checklist)

- **Encoding:** Save as UTF-8 CSV.
- **Quoting:** Any field with commas/quotes must be "quoted"; inner quotes escaped as "" (Excel does this automatically).
- **No blank rows:** They may be read as empty dicts; delete trailing empties.
- **Duplicates:** Avoid duplicate (probe,word) rows; last one wins if present.
- **Canonical tokens:** Decide early on lowercasing/lemmatization; Yoneda compares exact strings.
- **Small diffs:** After edits, re-run:
- python .\cthott_bench.py; python .\cthott_export.py; start .\out\bench_report.html

Inspect out\probe_matrix.csv and out\paraphrase_graph.dot/png.

With these schemas and practices, the bench stays **transparent, reproducible, and easy to extend**—you can grow from toy examples to richer mini-corpora without changing a line of metric code.

7. Implementation Details

Files and responsibilities

- **category.py** — *small categorical core*
    - Obj: nominal type tags (e.g., N, S, 1).
    - Morphism(name, dom, cod, fn): pure, total transformers.
    - compose(g,f), id_of(X): executable associativity/identity.
    - pullback(A, B, f, g): feature unification via fibered product; returns matching pairs plus projections.
      Used by: FunctorConsistency, PCS, Curry/Uncurry.
- **hott.py** — *homotopy & transport layer*
    - ParaphraseGraph: directed graph over sentences; add_path, neighbors.
    - simple_cycles(cap=2000): bounded DFS for cycles (deduplicated up to rotation).

- paraphrase_homotopy_index(): returns **PHI** and stores **PHI_nodes** (nontrivial cycles-per-node).
- transport_invariant(pairs, evaluate): equality of evaluator outputs across licensed substitutions.
  Used by: PHI/PHI_nodes, Transport.

- **metrics.py** — *metric helpers*
  - Probe(name, fn): boolean "use" predicate over words.
  - yoneda_probe_similarity(a,b,probes): mean match of probe truth tables.
  - pullback_coref_score(gold_links, pred_links): accuracy.
  - gluable(families): (#gluable, total) under simple conflict semantics.
    Used by: Yoneda*, PCS, Gluability.
- **cthott_bench.py** — *orchestrator (CSV → metrics → JSON)*
  - Loads CSVs from .\data\.
  - Computes all scores: **FunctorConsistency, YonedaSynonym, YonedaNonSyn, YonedaAUC, PCS, BCDefectNorm, Gluability, Transport, PHI, PHI_nodes, UA_witness, CurryParity, EffectLeakAudit**.
  - Prints a table and writes .\out\bench_report.json.
- **cthott_export.py** — *report & viz exporter (JSON → HTML/DOT/CSV)*
  - Reads bench_report.json.
  - Writes **.\out\bench_report.html**, **.\out\paraphrase_graph.dot**, and **.\out\probe_matrix.csv** (pivot of probe truths).
  - DOT renders to PNG via Graphviz (dot -Tpng …).
- **probe_sync.py** — *coverage tool (optional)*
  - Ensures every (probe,word) cell exists; fills missing with --fill {0|1} and rewrites probe_truth.csv sorted.
- **run_all.bat** — *one-click pipeline*
  - Activates venv → runs bench → runs exporter → (if Graphviz exists) emits PNG → opens HTML.

Complexity and runtime on a single PC

Let:

- P = #probes, K = #word pairs;
- M = #mentions, R = #pronouns;
- F = #discourse families, Kf = avg keys/family;
- T = #transport pairs, L = avg tokens/sentence;
- paraphrase graph: N nodes, E edges.

Metric costs (worst-case big-O):

- **FunctorConsistency:** O(1).
- **YonedaSynonym/NonSyn:** O(P·K).
- **YonedaAUC:** O(#pos · #neg) (on the scalar similarity scores; tiny in practice).
- **PCS:** for each pronoun, pullback-scan over mentions + sort ⇒ ~O(R·M log M) (often O(R·M)).
- **BCDefectNorm:** O(#scope_rows).
- **Gluability:** O(F·Kf).
- **Transport:** per sentence, normalize tokens and sort ⇒ O(T·L log L).
- **PHI / PHI_nodes:** cycle enumeration is exponential in theory; we *bound* DFS depth (≤8) and total cycles (cap=2000), so practical runtime ≈ O(N + E) on small graphs.
- **UA_witness, CurryParity, EffectLeakAudit:** O(1) to O(|X|·|A|) on tiny fixed sets.

Observed runtimes (your machine): all metrics + exports complete essentially instantly on toy data. On “small” sets (e.g., P≈100, K≈5–10k, N≈≤1k, E≈≤5k, T≈≤1k) expect seconds on a single Windows PC. Determinism: no RNG; iteration is key-sorted to stabilize outputs.

Exports: JSON report, HTML report, Graphviz DOT/PNG

- **.\out\bench_report.json** — single source of truth. Example:
- {
- "FunctorConsistency": 1.0,
- "YonedaSynonym": 1.0,
- "YonedaNonSyn": 0.667,
- "YonedaAUC": 1.0,
- "PCS": 1.0,
- "BCDefectNorm": 0.85,
- "Gluability": 0.667,
- "Transport": 1.0,
- "PHI": 0.25,
- "PHI_nodes": 0.286,
- "UA_witness": 1.0,
- "CurryParity": 1.0,
- "EffectLeakAudit": 1.0
- }
- **.\out\bench_report.html** — human-friendly summary with metadata and a command snippet to render the graph.
- **.\out\paraphrase_graph.dot** — Graphviz source. Render to PNG if Graphviz is installed:
- dot -Tpng .\out\paraphrase_graph.dot -o .\out\paraphrase_graph.png

Then open:

ii .\out\paraphrase_graph.png

- **.\out\probe_matrix.csv** — pivoted matrix (words × probes) for quick sanity checks and auditing.

Versioning tip: commit data/, out/bench_report.json, and (optionally) out/paraphrase_graph.dot. Diff JSON across commits to spot regressions.

Automation and PowerShell workflow

Everyday loop (interactive):

```
cd C:\Users\doug\aix\cthott

.\.venv\Scripts\Activate.ps1

python .\cthott_bench.py

python .\cthott_export.py

start .\out\bench_report.html
```

One-click run (batch):

```
.\run_all.bat
```

- Creates/activates the venv if missing.
- Runs bench → exporter → Graphviz (if available) → opens HTML.

Common helpers:

```
# Fill missing probe cells after adding words/probes

python .\probe_sync.py --fill 0


# View outputs and data

ii .\out

ii .\data

Get-Content .\out\bench_report.json
```

Troubleshooting quickies

- *Execution policy blocks venv activation:*
  Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
- *dot not found:*
  winget install Graphviz.Graphviz -s winget
  then (if needed) add to PATH:
  $env:Path += ";C:\Program Files\Graphviz\bin"

- *CSV quoting:* sentences with commas must be "quoted"; Excel handles this automatically.

Design invariants to preserve when extending

- Purity (metrics are pure functions of CSVs).
- Determinism (sort keys, fix traversal order).
- Side-effect discipline (effects via explicit monads/sets, not hidden state).
- Artifacts-first (JSON/DOT/CSV as first-class outputs for audit).

8. Results: Sample Runs & Sensitivity

Baseline scores on included toy data

Running the bench on the bundled CSVs (plus your small edits) yields the following representative snapshot:

- **FunctorConsistency 1.000** — executable associativity/identity hold on the toy morphism chain.
- **YonedaSynonym 1.000; YonedaNonSyn 0.667; YonedaAUC 1.000** — probe patterns perfectly separate positives from negatives in AUC, while one or two “hard negatives” still share some probe profile.
- **PCS 1.000** — pullback+tie-breaks select the gold antecedent for the demo pronouns.
- **BCDefectNorm 0.850** — cartesian (lawful) scope path is strongly preferred over the noncartesian alternative.
- **Gluability 0.667** — two of three discourse families glue; one is marked conflict: and rightly fails.
- **Transport 1.000** — the evaluator is invariant to the licensed active↔passive swaps in the toy pairs.
- **PHI 0.250; PHI_nodes 0.286** — the paraphrase graph contains a few **nontrivial** (length>2) cycles among mostly trivial 2-cycles.
- **UA_witness 1.000; CurryParity 1.000; EffectLeakAudit 1.000** — structural sanity checks pass on the baked-in micro-worlds.

These numbers are **deterministic**: with the same CSVs, you'll get the same JSON and DOT every run.

Sensitivity to added probes/pairs/edges

The bench is designed to move **predictably** when you add or alter data:

- **Adding probes** (new rows in probe_truth.csv):
    - If a new probe *agrees* on synonym pairs and *disagrees* on non-synonyms, **YonedaSynonym ↑**, **YonedaNonSyn ↑**, **AUC stays high**.
    - If a probe is noisy (e.g., always 1), it **dilutes** both signals (Synonym ↓, NonSyn ↓) but AUC may remain robust if enough informative probes remain.
    - Missing cells trigger coverage warnings or can be filled with probe_sync.py.
- **Adding synonym/non-synonym pairs** (synonyms.csv):
    - Clean positives (same probe patterns) push **YonedaSynonym ↑** and often keep **AUC ≈ 1**.
    - Hard negatives that accidentally match many probes (e.g., polysemous words) **lower YonedaNonSyn** and may reduce **AUC** if numerous.
- **Adding paraphrase edges** (paraphrases.csv):
    - Triangles/loops that are **distinct** (not just mirrored 2-cycles) increase **nontrivial cycles**, so **PHI** and **PHI_nodes** generally **rise**.
    - Adding parallel edges or near-duplicates mainly increases trivial 2-cycles, which can **lower PHI** (legacy metric) while **PHI_nodes** is more stable.
- **Changing scope scores** (scope.csv):
    - Raising the cartesian mean or lowering noncartesian increases **BCDefectNorm** smoothly; balanced means ≈ 0.5.
    - If the two sets invert (noncartesian > cartesian), the normalized score drops toward 0.
- **Coreference features** (mentions.csv, pronouns.csv):
    - More candidates with the same features increase reliance on tie-breakers (role/recency). Adjusting these visibly moves **PCS**.

- **Transport pairs** (transport.csv):
    - If you add licensed transformations that the evaluator does **not** normalize (e.g., deep quantifier alternations), **Transport** will drop, signaling the need for a stronger evaluator.

Visualizing paraphrase geometry (DOT/PNG)

The exporter writes out/paraphrase_graph.dot, which you rendered to PNG. Reading the picture:

- **Nodes** are sentences; **labeled edges** indicate paraphrase justifications (e.g., *active-passive*, *nominalization*).
- **Nontrivial cycles** (length > 2) show **higher-path** structure: multiple routes return to the same sentence via different transformations.
- Sparse graphs with only back-and-forth edges form many 2-cycles but few nontrivial loops—visually, these look like pairs with double arrows and no triangles or longer rings.

Diagnostic uses:

- If **PHI_nodes** is low, examine the DOT for **bridges** or **chains** with no closure; adding a justified edge to close a triangle typically raises the metric.
- If **PHI** drops while **PHI_nodes** holds steady, you're likely adding many trivial 2-cycles; consider deduplicating or weighting edges by reason.

Interpreting PHI vs. PHI_nodes

- **PHI (legacy)** = nontrivial / all cycles.
    - **Pros:** Sensitive to the *mix* of cycles; penalizes graphs swamped by trivial back-and-forth.
    - **Cons: Scale-sensitive**—as you add edges (including duplicates), the denominator (all cycles) can explode, pushing PHI down even when interesting loops exist.
- **PHI_nodes (recommended)** = nontrivial cycles / #nodes.
    - **Pros: Scale-free**, interpretable as "nontrivial loops per sentence." Adding unrelated nodes without new loops won't dilute the score.

- **Cons:** Doesn't penalize trivial 2-cycles directly (that's by design—use PHI or edge hygiene for that).

How to use both: Track **PHI_nodes** as your *primary growth* indicator when curating paraphrase structure; watch **PHI** as a *quality control* that alerts you when trivial edges dominate. A healthy graph sees **PHI_nodes** climb with curated cycles, while **PHI** stays flat or improves as you limit redundant edges.

Takeaway. The bench's metrics behave monotonically with the intuitions they're meant to capture. You can *steer* the numbers with principled edits: refine probe banks to shape Yoneda signals, add lawful edges to enrich homotopies, and strengthen evaluators where Transport flags gaps. Because every change is reflected in JSON and DOT, you can see **why** a number moved—not just **that** it did.

9. Case Studies

9.1 Synonym drift under probe changes

Setup. Start with probe_truth.csv containing four binary probes (heavy?, countable?, finance?, geo_or_metal?). The synonym bank pairs include *(rock, anvil)* (label 1) and non-synonyms *(bank, river)* (label 0). Baseline: **YonedaSynonym=1.00**, **YonedaNonSyn≈0.67**, **YonedaAUC=1.00**.

Manipulation. Add a noisy probe common_in_headlines? and (intentionally) mark it **1** for nearly every word. This new probe will agree for almost all pairs, pushing pairwise similarities up indiscriminately.

Outcome.

- **YonedaSynonym** may remain high (positives still agree), but
- **YonedaNonSyn** drops (negatives now also agree more often), and
- **YonedaAUC** can stay near **1.00** if enough informative probes remain, or decrease if the noise dominates.

Interpretation. The AUC is robust to a few bad probes but will flag systematic drift if noise overwhelms signal. In practice, you can (i) remove or down-weight weak probes, or (ii) expand the negative set with pairs that the noisy probe **should** separate (e.g., *(loan, river)* under finance?).

9.2 Coreference disambiguation via pullbacks

Setup. mentions.csv lists candidates:

Alice,f,sg,subj,2

Beth,f,sg,obj,3

Carlos,m,sg,subj,1

and pronouns.csv has:

she,f,sg,Alice

The **pullback filter** matches (gender, num); tie-breakers prefer role=subj, then larger recency.

Observation A (clear disambiguation). With the above data, both *Alice* and *Beth* match the pronoun's features, but the ranking selects *Alice* (subject preference), yielding **PCS=1.0**.

Observation B (recency takeover). If we flip recencies (Alice,…,recency=1; Beth,…,recency=5) while keeping role equal, the ranking returns *Beth*. If the gold stays *Alice*, **PCS** falls. This demonstrates that **pullbacks perform the hard constraint** and **ranking resolves multiplicity** explicitly—no hidden heuristics.

What to falsify. If adding a semantic feature (e.g., animacy) corrects false ties, PCS should **rise**. If PCS remains low, examine which feature is missing or mis-specified; the exported JSON and CSV make each miss auditable.

9.3 Scope illusions and BC-defect shifts

Setup. scope.csv stores paired scores for items where a lawful (cartesian) movement should commute (e.g., harmless QR), contrasted with a noncartesian movement (illicit reshuffling). Baseline:

cartesian,0.95

noncartesian,0.25

BCDefectNorm normalizes the gap (C − N) into [0,1].

Illusion test. Add items known to induce *every/each* illusions (e.g., "Every student read a paper" under a reading the system prefers but that conflicts with licensing). If your scoring function inadvertently favors the illicit path for these items (raising noncartesian or lowering cartesian), **BCDefectNorm** drops toward **0.5** or worse.

Outcome. After adding two illusion items with cartesian=0.55, noncartesian=0.45, you'll see the normalized score fall smoothly (e.g., **0.85 → 0.74**). This shows the metric can **quantify** how strongly your pipeline respects lawful scope, and where it bends under pressure.

Actionable. Tighten the evaluator (e.g., enrich quantifier typing or add licensing checks) and watch **BCDefectNorm** rebound.

9.4 Discourse (un)glueability in conflicting timelines

Setup. discourse.csv collects local constraints per mini-story:

1,topic,budget

1,tense,past

1,agent,committee

2,topic,budget

2,tense,conflict: past vs. present

2,agent,committee

Gluability is the fraction of families that admit a single global assignment consistent on overlaps. Family 1 glues; Family 2 explicitly marks a tense clash and fails.

Manipulation A (repair). Change Family 2 to:

2,tense,past

2,time_span,2018-2019

Now both topic and tense align; **Gluability** increases (e.g., **0.50 → 0.67**). The added time_span is harmless metadata and does not block gluing.

Manipulation B (masked conflict). If you represent tense by two keys (start_tense=past, end_tense=present) without a typed unifier, the current equality-based matcher may treat them as compatible, **inflating** Gluability. This is a deliberate *toy* assumption: to detect such cases, extend the unifier to intervals or lattice joins. The point is that **sheaf-style gluing exposes where your unifier is too permissive**.

Interpretation. Gluability is not a semantic "score" per se; it's a **well-posedness indicator**. If it falls as you add timelines drawn from real text, that signals the need for a richer overlap theory

(typed intervals, entity alignment), which you can plug in without changing the bench's outer loop.

Summary. Each case study shows **predictable, inspectable** shifts: probe edits drive Yoneda signals; feature and ranking choices drive PCS; lawful/illicit manipulations move BC-defect; timeline repairs lift Gluability. Because all changes are CSV-local and artifacts are exported (JSON, DOT/PNG), you can **trace every movement** back to a concrete, actionable edit.

10. Falsification Harness & Thresholds

Turning metrics into gates/alarms

Goal. Treat each score as a *specification check*. The harness reads out/bench_report.json, compares to a threshold profile, and returns **PASS/WARN/FAIL** with pointers to artifacts (HTML, DOT, CSV).

Gate types

- **Hard gates (must pass):** structural invariants that should never fail in a sane system.
  FunctorConsistency, UA_witness, CurryParity, EffectLeakAudit
- **Quality gates (numeric floor):** minimum acceptable behavior on curated data.
  YonedaSynonym, YonedaNonSyn, YonedaAUC, PCS, BCDefectNorm, Gluability, Transport
- **Trend gates (delta-based):** geometric/coverage signals that should *not regress* relative to the previous release.
  PHI_nodes (↑ or ≥ baseline), PHI (stable or ↑), *coverage warnings = 0*

Recommended thresholds (defaults)

- FunctorConsistency = 1.00 **HARD**
- UA_witness = 1.00 **HARD**
- CurryParity = 1.00 **HARD**
- EffectLeakAudit = 1.00 **HARD**
- YonedaAUC ≥ 0.90 ; YonedaSynonym ≥ 0.85 ; YonedaNonSyn ≥ 0.75
- PCS ≥ 0.80
- BCDefectNorm ≥ 0.70

- Gluability ≥ 0.60 *(require ≥3 families)*
- Transport ≥ 0.95
- PHI_nodes ≥ prior_release − 0.02 and **preferably** PHI_nodes ≥ 0.25 on toy set
- **Coverage warnings = 0** (any missing (probe,word) is **WARN**)

Threshold schema (example)

```
{
  "hard": ["FunctorConsistency","UA_witness","CurryParity","EffectLeakAudit"],
  "floors": {
    "YonedaAUC": 0.90, "YonedaSynonym": 0.85, "YonedaNonSyn": 0.75,
    "PCS": 0.80, "BCDefectNorm": 0.70, "Gluability": 0.60,
    "Transport": 0.95
  },
  "trend_min": { "PHI_nodes": -0.02 },
  "advisory": { "PHI": 0.20 },
  "require_min_families": 3
}
```

Pipeline integration (one liners)

- **Audit mode (no fail):** run bench → export → print report.
- **Enforcement mode:** compare JSON to thresholds and exit non-zero on **FAIL**; print **WARN** for coverage.

(We ship a tiny gate.py easily added later; current repo already emits all needed artifacts.)

Example failure modes and how they surface

1) Probe collapse (noisy or redundant probes)

- **Surface:** YonedaNonSyn falls, **coverage warnings** clean, **AUC** may hold briefly then declines as noise grows.
- **Artifact to inspect:** out/probe_matrix.csv (columns that are all 1s/0s).

- **Fix:** prune or down-weight flat probes; add discriminative probes; expand negatives that the probe should separate.

2) Polysemy leakage in pairs

- **Surface:** YonedaAUC drops while YonedaSynonym ≈ unchanged; non-syns share many probe values.
- **Artifact:** list of low-scoring negative pairs (add a per-pair dump—trivial to export).
- **Fix:** split senses or refine probes (e.g., finance? vs. riverine?).

3) Coreference ties misranked

- **Surface: PCS** falls despite correct pullback matches; many candidates tie on features.
- **Artifact:** mentions.csv showing identical (gender,num) and role/recency patterns.
- **Fix:** add features (person/animacy/case); adjust tie-breakers or provide a learned ranker **after** the pullback filter.

4) Scope evaluator bug

- **Surface:** BCDefectNorm drifts toward 0.5; cartesian and noncartesian means converge.
- **Artifact:** scope.csv; check stimuli and scoring consistency.
- **Fix:** tighten licensing (cartesian set) or scoring rubric; separate item types and track per-type BC.

5) Paraphrase edge explosion

- **Surface: PHI** drops while **PHI_nodes** steady—many trivial 2-cycles added.
- **Artifact:** paraphrase_graph.png shows dense double-arrows without triangles.
- **Fix:** deduplicate edges; weight by reason; curate cycles that close triangles/loops.

6) Transport evaluator too weak/too strong

- **Surface (weak):** Transport < 0.95 on obvious active↔passive;
  Surface (over-strong): Transport stays high even when pairs are *not* meaning preserving.
- **Artifact:** diff of evaluator outputs for failing pairs.
- **Fix:** upgrade canonicalizer (lemmas, typed roles) or restrict licensed transformations.

7) Structural violations

- **Surface:** any of FunctorConsistency, UA_witness, CurryParity, EffectLeakAudit < 1.0 → **HARD FAIL**.
- **Artifact:** console stack with example counter-evaluation (we log the exact term where equality failed).
- **Fix:** restore purity/normalization; repair witness functions.

Practical gating playbook

1. **Run in Audit first.** If any hard < 1.0 → stop and fix.
2. **Check coverage.** If missing cells > 0 → run probe_sync.py --fill 0; re-score; then hand-curate.
3. **Quality floors.** Any floor failed → inspect the specific CSV and exported artifacts noted above.
4. **Trend check.** Compare PHI_nodes to last release; regressions > 0.02 demand justification (waiver).
5. **Hysteresis.** Use a **grace band** ±0.01 around floors to avoid flapping on tiny edits; require two consecutive runs below floor to block CI.
6. **Waivers.** Allow a signed waivers.json listing (metric, reason, expiry date); CI prints waivers; none for **hard** gates.

Risk tiers

- **Red (block):** Hard gate fail; AUC < 0.85; Transport < 0.90.
- **Amber (investigate):** PHI_nodes − prior < −0.02; BCDefectNorm 0.60–0.70; Gluability 0.50–0.60.
- **Green:** all floors met; PHI_nodes stable or ↑; coverage 100%.

Bottom line. The harness turns abstract CT/HoTT desiderata into **actionable, enforceable** checks. Every fail points to a small CSV and a concrete artifact—so you can fix the dataset, the evaluator, or the witness function and immediately verify the improvement.

11. Validation & Calibration

Human judgments vs. YonedaAUC alignment (procedure)

Goal. Verify that "meaning-as-use" via probes agrees with human synonymy judgments, and calibrate the Yoneda signal to interpretable probabilities.

Data schema.

- synonyms_human.csv
  w1,w2,judge,score where score∈{0..4} (0=unrelated … 4=near-identical). At least 3 judges per pair.
- Optionally, map to a binary label per pair by majority (avg≥3 → 1 else 0) while **keeping** the mean score for correlation analyses.

Protocol.

1. **Sampling.** From synonyms.csv, draw a *stratified* set of pairs across the full Yoneda similarity range (e.g., 20% from each quintile). Add 10–20% "challenge" pairs (polysemes).
2. **Annotation.** Blind raters to probes. Provide 1–2 example sentences per word (if needed), but avoid definitional leakage. Target 3–5 raters; compute inter-annotator agreement (Krippendorff's α or Cohen's κ). Aim for **α ≥ 0.60**.
3. **Aggregation.** Per pair, compute:
   - mean human score H∈[0,4], std, and a binary label L by threshold (e.g., H≥3 → 1).
4. **Alignment metrics.**
   - **AUC(hum):** ROC-AUC of Yoneda similarity vs. L. Compare to **YonedaAUC** from the probe truth; they should match within sampling error.
   - **ρ/Spearman:** correlation between Yoneda similarity and H.
   - **Brier / ECE:** if you map Yoneda similarity s∈[0,1] to a probability, compute **Expected Calibration Error** by binning s and comparing binwise mean s to empirical positive rate.
5. **Calibration map.** Fit a simple logistic or isotonic regression $\hat{p} = \mathrm{Cal}(s)$ on 80% of annotated pairs; validate on the held-out 20%. Store Cal parameters to report *calibrated synonym probability* next to raw Yoneda scores.

Acceptance.

- AUC(hum) ≥ 0.90 on the curated sample; Spearman ρ ≥ 0.70.
- ECE ≤ 0.08 after calibration.
- If α<0.6, refine the rubric or examples; do **not** over-fit probes to noisy labels.

Probe curation loop.

- Plot **per-probe mutual information** with human labels; prune flat probes, add discriminative ones.
- Recompute YonedaAUC and ECE; stop when gains plateau.

Bootstrap/perturbation tests for metric stability

Why. We want tight confidence bands and to detect brittle metrics (those that swing under small, plausible changes).

A. Nonparametric bootstrap (confidence intervals)

For each metric m (YonedaAUC, YonedaSynonym, YonedaNonSyn, PCS, BCDefectNorm, Gluability, Transport, PHI, PHI_nodes):

1. **Unit of resample.**
    - Yoneda*: resample **pairs** with replacement, stratified by label.
    - PCS: resample **pronouns**.
    - BCDefectNorm: resample **items** within each cond.
    - Gluability: resample **families**.
    - Transport: resample **sentence pairs**.
    - PHI/PHI_nodes: resample **edges** (or nodes) with replacement.
2. **Iterations.** 1,000 bootstrap replicates; compute the metric each time.
3. **CI.** Report 2.5–97.5% quantiles (95% CI).
   Acceptance: CI width ≤ 0.03 for YonedaAUC on the current toy size; ≤ 0.05 for PCS/Transport; PHI_nodes may be wider on tiny graphs—use it comparatively (trend gate).

B. Jackknife / leave-one-feature-out (influence)

- **Probes:** recompute YonedaAUC leaving each probe out; rank probes by ΔAUC. Flag probes with negative contribution or outsized leverage (> 2× median Δ).
- **Coref features:** drop role or recency and observe ΔPCS; if Δ ≫ 0, you're relying on that feature—document it.
- **Paraphrase edges:** leave-one-edge-out Δ(PHI_nodes); list top-influential edges for manual review.

C. Noise injection (robustness)

- **Probe flips:** randomly flip p% of probe_truth cells (e.g., p∈{1,2,5}). Track YonedaAUC degradation curve; robust regimes degrade gracefully.
- **Scope jitter:** add Gaussian noise ε~N(0,σ^2) to score in scope.csv (clip to [0,1]); observe BCDefectNorm drift.
- **Graph perturbation:** degree-preserving edge swaps (rewire) as a null model; PHI_nodes under the null should be ≪ curated value. If not, your loops may be incidental.

D. Class/size sensitivity

- **Balance sweep:** vary positive:negative ratio (1:1 → 1:3) and recompute YonedaAUC/NonSyn/Synonym; AUC should be stable, means will shift (report both).
- **Scale sweep:** double the number of paraphrase edges with duplicates; PHI will likely drop (denominator inflation), PHI_nodes should remain steadier—this justifies reporting both.

E. Transport evaluator ablation

- Replace the canonicalizer with weaker/stronger variants (e.g., remove stopwording; add lemmatization; add dependency roles).
- Plot Transport vs. evaluator complexity; choose the *simplest* evaluator that clears the threshold with narrow bootstrap CIs.

Putting it together (calibration profile)

- **Publish** (in out/ or appendix):
    1. human-alignment table (AUC, ρ, ECE before/after calibration, α/κ),

2. bootstrap CIs for all metrics,
3. influence plots (leave-one-probe ΔAUC),
4. PHI_nodes under null graph perturbation vs. curated graph.

- **Set thresholds** using the lower bound of the 95% CI (e.g., **gate on LB(AUC) ≥ 0.90** rather than point estimate).
- **Freeze a probe set** once calibrated; subsequent edits run through the same validation harness.

Bottom line. Alignment tells us the metrics mean what we think; bootstrap and perturbations tell us they're **stable**. Together they turn the bench from a demo into a **measuring instrument**.

12. Reproducibility & Auditability

Deterministic runs, seed policy, and JSON artifacts

Determinism by design

- **No RNG, no gradients, no network.** Every metric is a pure function of CSV inputs.
- **Stable traversal order.** Lists, dicts, and graph nodes are **key-sorted** before iteration; cycle search is bounded (cap, depth ≤ 8) and seed-free.
- **Version-pinned tools.** Outputs depend only on: Python version, our .py files, CSV contents, and (optionally) Graphviz for PNG rendering.

Seed policy

- Current bench requires **no random seed**.
- If you add any randomized preprocessing, set both:
  - Python: import random, numpy as np; random.seed(0); np.random.seed(0)
  - Hash stability: PYTHONHASHSEED=0 (PowerShell for current process only: $env:PYTHONHASHSEED="0")

JSON is the source of truth

- out/bench_report.json is the canonical artifact—**machines diff this**.
- It's small, human-readable, and round-trippable; exporters (HTML, DOT) are *derived* views.

Recommended metadata (lightweight)
Augment bench_report.json with a top-level "meta" block:

- run_time: ISO timestamp
- python: sys.version
- bench_version: short git SHA of the repo
- data_checksums: SHA-256 per CSV in data/
- host: OS + platform
  This makes each run *standalone verifiable* without external state.

(If you want, we can add this to cthott_bench.py in one pass later.)

Diff-based change tracking across versions

What to diff

- **Primary:** out/bench_report.json → numeric deltas per metric.
- **Structure:** out/paraphrase_graph.dot → added/removed nodes/edges.
- **Data:** data/*.csv → row-level additions/removals/edits.

Human diffs (Windows)

JSON numeric diff (quick glance)

Get-Content .\out\bench_report.json

File compare (semantic inspection)

fc .\out\bench_report.json ..\prev\out\bench_report.json

DOT/CSV textual diffs

git diff -- .\out\paraphrase_graph.dot

git diff -- .\data

Semantic diff (tolerant)
Use a tiny helper to highlight *meaningful* changes ($>|\Delta| \geq \varepsilon$):

```
@'
import json, sys
from math import isfinite
EPS = 1e-3
a = json.load(open(sys.argv[1], "r", encoding="utf-8"))
b = json.load(open(sys.argv[2], "r", encoding="utf-8"))
def flat(d): return {k:v for k,v in d.items() if isinstance(v,(int,float))}
da, db = flat(a), flat(b)
keys = sorted(set(da)|set(db))
print("Metric, old, new, delta")
for k in keys:
    va, vb = da.get(k, float("nan")), db.get(k, float("nan"))
    if all(isinstance(x,(int,float)) and isfinite(x) for x in (va,vb)):
        d = vb - va
        if abs(d) >= EPS:
            print(f"{k},{va:.3f},{vb:.3f},{d:+.3f}")
'@ | Set-Content -Encoding UTF8 .\diff_report.py


# usage:
python .\diff_report.py .\out\bench_report.json ..\prev\out\bench_report.json
```

Version control playbook

- Commit **both** data/ and out/bench_report.json.
- Tag releases (v0.3.0) so you can compare json across tags:
- git diff v0.2.0:vect/out/bench_report.json v0.3.0:vect/out/bench_report.json
- Keep a CHANGELOG.md summarizing what changed in **data** and **code** for each version.

Reproducibility bundle (zip and share)

```
# capture environment
pip freeze > .\out\env.txt
$stamp = (Get-Date).ToString("yyyyMMdd_HHmmss")
Compress-Archive -Force -Path `
  .\category.py, .\hott.py, .\metrics.py, .\cthott_bench.py, .\cthott_export.py, `
  .\data\*, .\out\bench_report.json, .\out\env.txt `
  -DestinationPath ".\ai1-linguo_run_$stamp.zip"
```

Anyone can unzip and **re-run** to confirm the numbers (HTML/DOT regenerate identically).

Audit trail checklist

- ✅ Deterministic code paths (no hidden state or RNG)
- ✅ Sorted traversals and bounded cycle search
- ✅ JSON report with metadata
- ✅ CSV checksums and Git tags for provenance
- ✅ Diff scripts for numeric and structural changes
- ✅ Repro bundle including env.txt (package versions)

How auditors work with this

1. Verify SHA of CSVs matches data_checksums.
2. Re-run cthott_bench.py → compare JSON to submitted report (must match).
3. Render DOT → scan for added/removed cycles consistent with PHI/PHI_nodes deltas.
4. Spot-check data edits in Git history (e.g., probe additions vs. YonedaAUC shifts).

Bottom line. By elevating bench_report.json to the single source of truth, fixing iteration order, and keeping tiny diffs first-class, the bench produces **repeatable numbers with a legible paper trail**. This turns results from “screenshots” into **verifiable claims**.

13. Comparison to Learning-Centric Systems

Where ai1-Linguo fits

- **Role: Verifier/curator**, not a learner. ai1-Linguo turns CT/HoTT desiderata into *tests* that any system (rule-based, neural, or hybrid) must satisfy on small, auditable datasets.
- **Granularity:** Operates at **law level** (associativity, scope commutation, pullback agreement, transport invariance, homotopy structure) rather than at token-prediction level.
- **Deliverable:** Deterministic **artifacts** (JSON, DOT/PNG, HTML) that make structural claims falsifiable. This complements learning systems whose outputs are stochastic and scale-dependent.

Why learning alone is insufficient

- **Objective mismatch:** Minimizing loss ≠ obeying laws. A model can fit data and still violate functoriality, swap scope illicitly, or leak presuppositions.
- **Opacity:** LLMs offer few *mechanistic* guarantees. The bench surfaces **where** and **how** structure breaks—down to the exact CSV row or graph loop.
- **Regression risk:** Model updates can silently degrade compositional behaviors; local, deterministic gates catch this early.

Hybrid flows: LLM as data generator, bench as verifier

A) Paraphrase harvesting (grows PHI_nodes without noise)

1. **Generate:** Ask an LLM for paraphrase candidates of seed sentences, with a **reason tag** (e.g., “active-passive”, “nominalization”).
2. **Insert:** Append to paraphrases.csv as (s1,s2,reason).
3. **Verify:** Run bench → accept edges that (a) **preserve Transport** on your evaluator and (b) **increase PHI_nodes** (nontrivial cycles per node).
4. **Curate:** Reject edges that mostly add trivial 2-cycles (PHI drops while PHI_nodes flat).

Outcome: A paraphrase graph whose **geometry** reflects real alternations, not duplicates.

B) Probe induction for synonyms (stabilizes YonedaAUC)

1. **Cluster:** Use an embedding model to cluster words (optional).
2. **Propose probes:** Prompt the LLM for *binary "use" probes* that separate clusters (e.g., finance?, countable?, animacy?).
3. **Annotate:** Autolabel a draft matrix for probe_truth.csv; **human-check** a stratified subset.
4. **Select:** Compute **YonedaAUC**, leave-one-probe ΔAUC (influence). Keep probes with positive contribution; drop flat/noisy ones.
5. **Sync:** Run probe_sync.py to fill any missing cells; re-export probe_matrix.csv for inspection.

Outcome: A compact, discriminative probe bank aligned with **human judgments** (cf. §11).

C) Coreference curation via pullbacks (PCS ↑ with features)

1. **Detect mentions:** Use an LLM or a tagger to propose mentions + features (gender/number/animacy/role/recency).
2. **Gold links:** Collect minimal gold pronoun→antecedent pairs.
3. **Verify:** Pullback filters candidates; ranking (role/recency) breaks ties; **PCS** reports accuracy.
4. **Iterate:** When PCS fails, add the missing **feature** rather than ad-hoc heuristics.

Outcome: A transparent, feature-complete coref mini-set where errors are *explainable*.

D) Scope minimal pairs (BC-defect as commutation fitness)

1. **Synthesize:** Have an LLM generate controlled minimal pairs that differ only by a scope move.
2. **Score:** Put cartesian vs noncartesian items into scope.csv.
3. **Gate:** Track **BCDefectNorm**; prefer items/models that widen the lawful–illicit gap.

Outcome: A targeted battery for lawful scope; easy to expand while keeping semantics legible.

How to use the bench *with* learners

Training-time (soft integration)

- **Auxiliary objectives:** When training smaller task-specific models, add penalties for **Transport** failures, encourage higher **YonedaAUC**, and penalize **BC-defect**. (Treat bench scores as soft constraints; keep final verification hard.)
- **Curriculum:** Promote samples that **improve** PHI_nodes or reduce BC-defect into the training set first.

Post-training (hard verification)

- **Model selection:** For multiple checkpoints, pick the one that clears all **hard gates** and maximizes quality floors (cf. §10).
- **Release CI:** On each model update, run the bench locally; block release on any **hard** fail, and require waivers for quality/trend regressions.

Data lifecycle

- **Generator → Verifier → Library.** Anything that passes gates is added to your **curated CSVs**. Future models are evaluated on this growing, structured library—turning ad-hoc datasets into a **measuring instrument**.

Practical integration points

- **Interfaces the generator must satisfy**
    - **Paraphrases:** emit (s1,s2,reason) with reasons from a controlled vocabulary.
    - **Probes:** emit candidate binary probes with **definitions** and example decisions.
    - **Coref:** emit mentions with *typed features*; pronouns with gold antecedents for a subset.
    - **Scope:** emit labeled minimal pairs with a short description of the intended lawful move.
- **Verifier feedback loop**
    - Accept if **Transport=1** on the pair and **PHI_nodes** increases; otherwise reject with a reason (“adds only a 2-cycle”).

- Accept a probe if **ΔAUC > 0** and it's not redundant (MI/ΔAUC under leave-one-probe jackknife).
- Log all accept/reject decisions to a small **provenance** column in the CSVs (who/when/why).

Strengths & limitations vs. LLM-only pipelines

Strengths

- **Structure-aware:** catches compositional and logical bugs invisible to accuracy.
- **Local-first & reproducible:** runs offline; artifacts are small and auditable.
- **Data hygiene:** curates datasets that *express laws*, not just labels.

Limitations

- **Scale:** the bench is **small-N by design**; it verifies, it doesn't train.
- **Evaluator dependence:** Transport and BC rely on the chosen evaluator/rubric; upgrade them as your target phenomena grow.
- **Coverage:** PHI/PHI_nodes reflect the graph you curate; they're only as rich as your edges.

Mental model: "The model proposes; the mathematics disposes."

Use the LLM for breadth—creative paraphrases, probe ideas, candidate features. Use ai1-Linguo to **accept only what is structurally consistent**, leaving a crisp trail (JSON/DOT/CSV) that anyone can re-run on a single PC.

14. Limitations & Threats to Validity

Toy scale

Threat. The bench is intentionally small-N: cycle search is depth-bounded, coref worlds are tiny, scope items are minimal pairs. Strong scores on toy data may not transfer to open-domain corpora.
Mitigations. (i) Treat results as **instrument calibration**, not global claims; (ii) report bootstrap CIs (§11) and publish the exact CSVs; (iii) scale by *adding families*, not by loosening laws—keep

determinism and artifacts intact; (iv) document runtime caps (cycle depth, cap) alongside conclusions.

Probe subjectivity

Threat. Probes in probe_truth.csv ("meaning-as-use") are author-defined and binary; different curators may disagree on what a probe tests or how a polysemous word should score.
Mitigations. (i) Human alignment study with inter-annotator agreement and calibration (§11); (ii) leave-one-probe influence (ΔAUC) to detect over-leveraged probes; (iii) provenance fields (source, annotator, date, license) and a **probe rationale** column; (iv) sense-splitting guidance when a word straddles incompatible uses.

Dataset bias

Threat. Synonym/non-syn pairs, scope items, and paraphrase edges may reflect curator biases (domain, register, template effects). Graph structure can be shaped by convenience (many active↔passive pairs) rather than true alternation diversity.
Mitigations. (i) Stratified sampling across parts of speech, domains, and constructions; (ii) adversarial additions (hard negatives, rare alternations); (iii) null-model checks (degree-preserving graph rewiring for PHI_nodes) and balance sweeps for Yoneda metrics (§11); (iv) release a **manifest** and changelog noting each addition's motivation.

Over-interpretation risks

Threat. Reading compliance scores as "model intelligence" or "semantic competence" is a category error. A system can ace our gates yet fail at tasks outside the bench's scope; conversely, a useful system may fail a gate because our evaluator is too weak (e.g., Transport canonicalizer).
Mitigations. (i) Frame scores as **lawful-behavior indicators** on **these** datasets; (ii) pair every claim with artifacts (JSON, DOT, matrix); (iii) include **counterexamples** when a metric is high but known failures exist; (iv) maintain **hard vs. quality** gates (§10) to avoid collapsing all nuance into a single "pass."

Scope of CT/HoTT idealizations

Threat. Our constructions are simplified instantiations: pullbacks equate features by string equality; gluing treats conflict: as a hard clash; BC-defect reduces commutation to paired scalars; univalence is witnessed on finite samples; effects model only presuppositions. Real language has gradience, defeasibility, and world knowledge these abstractions don't capture.
Mitigations. (i) Make the **idealization boundary** explicit in each section; (ii) provide plug-points for typed unification (intervals, lattices), richer evaluators (dependency roles for Transport), graded probes, and larger witness sets; (iii) add **falsification hooks**: if strengthening the unifier

reduces Gluability as expected, our previous score was over-permissive; (iv) flag any claim that depends on a toy evaluator with a visible footnote ("evaluator-dependent").

Metric artifacts & brittleness

Threat.

- **PHI vs. PHI_nodes.** PHI is denominator-sensitive (many 2-cycles depress it); PHI_nodes ignores trivial 2-cycles and may miss low-quality duplication.
- **AUC stability.** With few pairs, AUC can be deceptively perfect.
- **PCS tie-breakers.** Heuristics (role, recency) dominate when features are sparse.
  Mitigations. (i) Always report **both** PHI and PHI_nodes and show the DOT; (ii) publish bootstrap CIs and leave-one-unit influence; (iii) enrich features before adding more heuristics; (iv) cap and log cycle search so missed long loops are an acknowledged limitation, not a hidden one.

External validity & construct validity

Threat. High Transport may reflect an over-aggressive canonicalizer; high BCDefectNorm may reflect item selection rather than lawful reasoning; Yoneda success may reflect probe leakage.
Mitigations. (i) Ablations: weaken/strengthen evaluators and observe monotone changes (§11E); (ii) preregister thresholds and item templates; (iii) cross-dataset validation (hold-out sets with different authors); (iv) blinded human studies for synonyms to check construct alignment.

Ethical/operational cautions

Threat. "Compliance" language can be misused as a quality badge; deterministic numbers may invite Goodharting.
Mitigations. (i) Treat gates as **safeguards**, not scores to maximize blindly; (ii) log all dataset edits with rationale; (iii) require waivers for threshold overrides with expiry (§10); (iv) prefer publishing **counterexamples** along with strong scores.

What would falsify us (explicit)

- A probe set passing YonedaAUC ≥ 0.9 yet **disagreeing** with blinded human judgments (AUC_hum ≪ 0.8).
- Transport staying high while a stronger evaluator finds systematic mismatches on the same pairs.
- PHI_nodes comparable to degree-preserving null graphs.

- BCDefectNorm not improving when lawful constructions are strengthened.
  Any of these would trigger a **spec revision** (evaluator upgrade, probe pruning, or metric redesign), documented in the changelog.

Bottom line. ai1-Linguo is a **measuring instrument**, not a theory of everything: useful precisely because its assumptions, limits, and artifacts are explicit and testable.

15. Ethical Considerations

Transparency, interpretability, and responsible evaluation

- **Artifacts-first by design.** Every claim comes with *inspectable* outputs: bench_report.json (numbers), paraphrase_graph.dot/png (structure), and probe_matrix.csv (the "why"). These are meant to be published with the paper/report so others can reproduce and audit.
- **Method disclosure.** For each metric, report (i) data sources and licensing, (ii) evaluator assumptions (e.g., the Transport canonicalizer), (iii) limits/caps (cycle depth, search cap), and (iv) thresholds and waivers used (§10). Make evaluator-dependent claims explicit.
- **Counterexamples alongside scores.** High scores without failure cases invite overreach. Include small tables of known misses (e.g., paraphrase pairs that violate Transport; scope items that narrow the BC gap only slightly).
- **Human annotation ethics.** When collecting judgments for calibration (§11), provide a clear rubric, track inter-annotator agreement, and compensate fairly. Avoid tasks likely to elicit sensitive inferences; document instructions and de-biasing steps.
- **Privacy and PII.** Keep CSVs free of PII. If you must include real snippets, anonymize and verify license/consent. Prefer synthetic or public-domain text; store any restricted text outside the repo and reference via IDs.
- **Bias awareness.** Probes can encode stereotypes (e.g., gendered roles). Use balanced examples, stratified sampling, and bias checks (e.g., swap roles/entities and re-measure). Document observed asymmetries; don't bury them in aggregates.
- **Accessibility & verifiability.** Keep the bench small and local-first so external researchers—including those without large compute—can replicate results. Provide a reproducibility zip (code, CSVs, JSON, env file) and a one-line run command.

Avoiding misuse of "compliance" scores

- **What the scores are—and are not.**
    - **Are:** *law-specific fit* on a *small, declared dataset* under explicit evaluators.
    - **Are not:** general intelligence, broad semantic competence, or safety guarantees.
- **No single-number marketing.** Do *not* average the battery into one "compliance" score. Keep metrics separate; each measures a different structural property with different threat models.
- **Goodhart safeguards.** If thresholds become targets, teams may "teach to the test" (e.g., overfitting probes or adding trivial edges to game PHI). Mitigate with:
    - **Trend gates** (watch deltas, not just absolutes),
    - **Bootstrap CIs** (flag brittle wins),
    - **Null models** (e.g., graph rewiring baselines),
    - **Adversarial sets** (hard negatives and counter-cases).
- **Evaluator dependence disclaimers.** Transport/BC outcomes depend on chosen evaluators. Never export scores without naming the evaluator and its version; treat evaluator swaps as spec changes, not silent upgrades.
- **Scope-bound claims.** Frame results as: "On dataset D with evaluator E, metric M achieved X±CI." Avoid general claims unless validated on independent sets (§11).
- **Gating & decisions.** The bench should **not** be used as the sole criterion for high-stakes decisions (hiring, admissions, credit, etc.). It's a *research instrument* and a *CI guardrail*, not a compliance stamp for people-facing deployments.
- **Waivers with expiry.** When thresholds are not met but you proceed, require a written waiver stating the risk, rationale, and an expiration date. Publish waivers with the report.
- **Open record of changes.** Maintain a changelog and dataset manifest (provenance, licenses, edits). Retroactive edits that improve scores must be documented and re-validated; don't overwrite history.

Bottom line. ai1-Linguo is built to *encourage* ethical practice: small, local, auditable, and explicit about assumptions. Used responsibly, it helps teams find and fix structural weaknesses. Misused as a proxy for global competence, it risks overclaiming and Goodharting. Keep artifacts public, claims bounded, and evaluators named—then the numbers can do honest work.

16. Future Work

Larger probe banks; richer sheaf models; higher-category moves

16.1 Larger probe banks (scalable, calibrated "meaning-as-use")

What. Grow probe_truth.csv from dozens → hundreds of binary probes with clear rubrics and calibration to human judgments.

How.

- **Authoring loop.** (i) prompt LLMs for *candidate* probes with rationales; (ii) curate a stratified subset with human raters; (iii) run §11 alignment (AUC/ρ/ECE) and keep probes with positive ΔAUC under leave-one-probe ablation.
- **Schema bump.** Allow graded values: value ∈ {0, 0.5, 1} (or [0,1]) and add a simple threshold during Yoneda similarity, e.g. match = $1\{|va-vb| \le \tau\}$ with τ configurable per probe family.
- **Coverage automation.** Extend probe_sync.py to (a) prefill with *unknown* rather than 0, (b) emit a **coverage matrix heat map** in the exporter.
- **Probe governance.** Add columns rationale, examples, owner, review_date; refuse probes without a one-line operational definition.

Expected gains. Higher **YonedaAUC** on held-out human labels; stability under bootstrap; clearer failure localization (per-probe diagnostics).

16.2 Richer sheaf models (typed unification, partial gluing)

What. Replace string-equality gluing with typed unification on overlaps.

How.

- **Typed unifier module** (unify.py):
  - **Time**: intervals with overlap/containment;
  - **Entity roles**: lattice join for agent/patient;
  - **Tense/aspect**: finite algebra with inconsistency tags.
- **Partial sections.** Let families glue **partially** when only some keys conflict; report both a binary **Gluability** and a **glued-mass ratio** (fraction of keys that unify).

- **Conflict explanations.** Export a per-family CSV of (key, v1, v2, reason) for non-gluing causes; link it from HTML.

Expected gains. Fewer false glues; actionable repair traces; a second metric (GluedMass) sensitive to near-misses.

16.3 Higher-category moves (2-cells, profunctors, pushouts)

What. Move beyond 1-cells:

- **2-cells / rewrites between paraphrases.** Attach witnesses (e.g., "active→passive" then "dative-shift") as composable 2-morphisms; check **naturality squares** for common transformations.
- **Profunctors / relations.** Treat predicate-argument structure as a profunctor; test simple **Beck–Chevalley-like** conditions at the relational level.
- **Pushouts** and **β-laws.** Add a toy pushout check where two sub-sentences are glued along a shared interface; verify the β equalities (u∘inl = …, u∘inr = …) hold for your constructors.

How.

- Add two_cell.py with a Rewrite(name, src, tgt, witness) class and checkers for square commutation.
- Extend the DOT exporter to render *2-cell labels* on "faces" by duplicating paths with style hints.

Expected gains. Expressivity for multi-step paraphrase reasoning; finer BC-style diagnostics that include actual transformation paths.

Integration with ai1 term-rewriting engine and GUI

16.4 ai1 ↔ Linguo coupling (equational witnesses)

What. Feed the bench with **machine-checkable witnesses** from ai1's typed rewriting to normal forms (NF).

How.

- **Witness API.** Define a tiny JSON (out/witnesses/*.json) produced by ai1:
- {"claim": "transport_eq", "left": "s1", "right": "s2",
- "nf_left": "...", "nf_right": "...", "steps": ["ruleX", "ruleY"], "verdict": true}
- **New metrics.**
    - **β/η Witness Rate**: fraction of claims with verdict=true and nf_left= nf_right.
    - **Mid-Identity Elimination**: measured rate of g∘id∘f → g∘f eliminations in traces.
- **Trace viewer.** Add a panel in the HTML report that collapses/expands a witness trace for failing cases.

Expected gains. "Why" attached to "what"—every equality used by a metric can be clicked through to its derivation.

16.5 Lightweight GUI

What. A simple local UI to edit CSVs, run the bench, and visualize graphs.

How.

- **Option A (no extra deps):** Tkinter: file picker + run button + embedded webview for bench_report.html.
- **Option B (quick dev):** Streamlit/Gradio for an in-browser experience (installable; preserves local-first model).
- **Live PHI preview.** As you add an edge in a form, recompute PHI/PHI_nodes and redraw a minimized graph snippet.

Expected gains. Low-friction curation; faster iterate-and-check loops for non-programmers.

Community datasets and plug-in metrics

16.6 Community datasets (manifests, licenses, splits)

What. A shared repo of mini-batteries with manifests.

How.

- **Dataset card** (data/manifest.json) required: name, version, license, provenance, authors, contact.
- **Splits.** Provide train/dev/test CSV folders to separate **curation** from **evaluation**.
- **Linting.** Add lint_data.py to check quoting, duplicates, coverage, and manifest presence.

Governance. PR template asks for: license confirmation, probe rationales, and a diff of bench_report.json + PHI DOT png.

16.7 Plug-in metrics (extensibility API)

What. Let others drop in a metric without editing the core.

How.

- **Plugin contract** (plugins/<name>/metric.py):
- # must return {"key": float in [0,1]}, optional "artifacts" dict
- def compute(data_dir: str, out_dir: str) -> dict: ...
- NAME = "MyMetric"
- ORDER = 42 # placement in the HTML table
- **Loader.** cthott_bench.py scans plugins/*/metric.py, imports safely, executes compute, merges keys, and writes any artifacts under out/plugins/<name>/.
- **Exporter hook.** cthott_export.py renders plugin metrics in a dedicated section with links to their artifacts.

Safety. Keep plugins pure (no net), time-cap execution, and sandbox imports to stdlib + repo modules.

16.8 Bench-as-service (still local)

What. A thin CLI that other tools call.

How.

- python cthott_bench.py --json out/bench_report.json --quiet
- Non-zero exit on gate failures (reads thresholds.json).
- Optional --diff prev_report.json to print a delta table (uses diff_report.py logic).

Expected gains. Easy CI integration for teams using any generator (LLM or rules).

Milestones (suggested)

- **M1 (4–6 weeks):** Typed unifier v1 + partial gluing; plugin loader; ΔAUC probe ablation tooling.
- **M2 (6–8 weeks):** ai1 witness API + β/η Witness Rate; HTML trace viewer.
- **M3 (8–12 weeks):** 2-cell rewrites + naturality checks; PHI basis export (independent nontrivial cycles).
- **M4 (ongoing):** Community datasets (two external contributors), linter + manifest policy, Streamlit/Tk UI prototype.

Risks & mitigations

- **Overfitting to probes/edges.** Use held-out human sets and null-graph baselines (§11).
- **Complexity creep.** Keep new logic in *modules* (unify, two_cell, plugins) and preserve the deterministic, artifact-first core.
- **Data licensing.** Enforce manifest + license fields; allow hashed IDs for restricted text.

North star. Keep ai1-Linguo **small, local, and legible**, while widening its reach: more probes, richer gluing, explicit higher-structure—and a clean socket for community metrics and ai1's equational witnesses.

17. Conclusion

Summary: what executable CT/HoTT brings to language evaluation

ai1-Linguo reframes language evaluation around **equations, structure, and witnesses** rather than loss curves. By compiling a small palette of CT/HoTT constructions into runnable checks—**functoriality** (associativity/identity), **pullbacks** for agreement, **Beck–Chevalley** for lawful scope, **sheaf gluing** for discourse, **homotopies** for paraphrase cycles, **univalence** for concept identification, **curry–uncurry** for argument structure, and **monads** for presuppositions—the bench turns abstract desiderata into **deterministic metrics** with **machine-checkable artifacts**. Each score is backed by concrete CSV inputs, a JSON report, and (where appropriate) a DOT/PNG visualization, so deviations are **falsifiable** and **auditable**.

Practically, this yields four benefits:

- **Law-level guarantees.** We check whether transformations preserve the equations they ought to, not merely whether a prediction matches a label.
- **Traceability.** Failures point to a row, a loop, or a witness—problems are repairable, not mystical.
- **Determinism & locality.** Everything runs offline on a single PC; identical inputs reproduce identical numbers.
- **Complementarity.** Learning systems can propose paraphrases, probes, and features; the bench **verifies** structural sanity and curates durable datasets.

The result is a compact instrument that elevates **explanations to first-class artifacts** while keeping the workflow small, legible, and testable.

Invitation to reproduce and extend

Reproducing the paper's results is deliberately simple: place the provided files in a folder, run the bench, and inspect the exported **HTML**, **JSON**, and **DOT/PNG**. If your numbers differ, the JSON and graphs make the discrepancy actionable.

We invite you to extend the bench in three ways:

1. **Grow the data.** Add probes (probe_truth.csv), synonym/non-synonym pairs (synonyms.csv), paraphrase edges (paraphrases.csv), scope items (scope.csv), and discourse families (discourse.csv). Use probe_sync.py to maintain coverage, and watch how **YonedaAUC**, **BCDefectNorm**, **PHI/PHI_nodes**, and **Gluability** move.

2. **Enrich the semantics.** Swap in stronger evaluators (e.g., typed canonicalizers for Transport; interval unifiers for gluing). Add higher-category moves (2-cells, pushouts) or ai1 rewriting **witnesses** that justify equalities step-by-step.
3. **Plug in metrics.** Contribute a small, pure function that consumes CSVs and returns a 0,10,10,1 score plus artifacts; let the exporter display it alongside the core battery.

Keep changes documented with manifests and diffs; pair strong scores with counterexamples; and treat thresholds as gates, not trophies. If many hands add small, well-licensed CSVs and clear checks, we'll accumulate a shared, **executable commons** for compositional semantics—one that anyone can run, verify, and build upon on an ordinary PC.

18. Availability & How-To

Folder layout (what each file does)

```
cthott\
 ├– category.py          # Objects, morphisms, compose(), id_of(), pullback()
 ├– hott.py              # ParaphraseGraph, PHI/PHI_nodes, transport_invariant()
 ├– metrics.py           # Probe class, Yoneda similarity, PCS, gluable()
 ├– cthott_bench.py      # Orchestrator: CSV → metrics → JSON (report)
 ├– cthott_export.py     # Exporter: JSON → HTML, DOT, probe_matrix.csv
 ├– probe_sync.py        # Coverage tool: fill missing (probe,word) cells
 ├– run_all.bat          # One-click runner (bench → export → PNG → open)
 ├– data\                # All editable CSVs (your “dataset”)
|  ├– probe_truth.csv    # probe,word,value
|  ├– synonyms.csv       # w1,w2,label
|  ├– mentions.csv       # id,gender,num,role,recency
|  ├– pronouns.csv       # form,gender,num,gold
|  ├– paraphrases.csv    # s1,s2,reason
|  ├– discourse.csv      # family,key,value
|  ├– scope.csv          # cond,score
|  └– transport.csv      # s1,s2
└– out\                 # Auto-generated artifacts (do not hand-edit)
   ├– bench_report.json  # Canonical scores (source of truth)
   ├– bench_report.html  # Human-readable summary
   ├– paraphrase_graph.dot  # Graphviz source (render to PNG if you like)
   ├– paraphrase_graph.png  # (optional) Graphviz render
   └– probe_matrix.csv   # Pivoted matrix (words × probes)
```

Run commands (Windows PowerShell)

First run

```
cd C:\Users\doug\aix\cthott

py -3 -m venv .venv

.\.venv\Scripts\Activate.ps1

python .\cthott_bench.py

python .\cthott_export.py

start .\out\bench_report.html
```

One-click thereafter

```
.\run_all.bat
```

(Optional) Render paraphrase graph to PNG

```
# Install once if not present:

winget install Graphviz.Graphviz -s winget

$env:Path += ";C:\Program Files\Graphviz\bin"


dot -Tpng .\out\paraphrase_graph.dot -o .\out\paraphrase_graph.png

ii .\out\paraphrase_graph.png
```

Keep probe coverage complete after edits

```
.\.venv\Scripts\Activate.ps1

python .\probe_sync.py --fill 0

python .\cthott_bench.py; python .\cthott_export.py; start .\out\bench_report.html
```

Quick "How-To" tasks

- **Add a synonym pair:** append a row to data\synonyms.csv, then ensure both words have entries for **every** probe in probe_truth.csv (or run probe_sync.py as a stopgap).
- **Add a probe:** append rows to probe_truth.csv for **every** word you care about; each row is probe,word,value.

- **Add a paraphrase:** add "s1","s2","reason" to paraphrases.csv (quotes required if commas); re-run to update DOT/PNG and **PHI/PHI_nodes**.
- **Tighten scope items:** add paired rows in scope.csv labeled cartesian vs noncartesian; watch **BCDefectNorm** move.
- **Tweak coreference:** add mentions/pronouns in their CSVs; watch **PCS**.

Troubleshooting checklist

Environment / execution

- **Activation blocked:**
  Set-ExecutionPolicy -Scope Process Bypass then re-run .\.venv\Scripts\Activate.ps1.
- **Multiple Python installs:**
  Use py -3 to create the venv; verify with .\.venv\Scripts\python.exe --version.
- **Graphviz "dot" not found:**
  winget install Graphviz.Graphviz -s winget then
  $env:Path += ";C:\Program Files\Graphviz\bin".

Data sanity

- **CSV quoting:** Any field containing commas or quotes must be "quoted" with inner quotes doubled (""). Excel does this for you.
- **Encoding:** Save CSVs as **UTF-8**. Avoid blank trailing lines.
- **Coverage warnings (Yoneda):** Run python .\probe_sync.py --fill 0 and then hand-curate the new cells (don't leave them all zeros long-term).
- **Duplicate rows:** Avoid duplicate (probe,word) entries—last one wins. Clean with a quick sort/unique in Excel or a script.

Output differences / regressions

- **Numbers changed unexpectedly:**
  - Confirm you are in the intended folder (pwd), and re-run python .\cthott_bench.py.
  - Open out\bench_report.json and compare to an earlier copy (use git diff or the provided diff_report.py from §12).

    - Check whether CSVs changed (Git status) or if you altered an evaluator (e.g., transport canonicalizer).

- **PHI drops while PHI_nodes steady:** You likely added many trivial 2-cycles. De-duplicate edges or add edges that close triangles.
- **PCS fell after edits:** Inspect mentions.csv—do multiple candidates share identical features? Add discriminative features (person, animacy) or adjust tie-breakers.

Common mistakes & fixes

- **Here-strings (@'... '@) pasted into Python files:** Ensure those only appear in **PowerShell** when creating files, not inside the .py source. If you see @' in a .py, replace the file with the clean version.
- **Transport too forgiving/strict:** Swap in a stronger/weaker evaluator in hott.py (e.g., add lemmatization or role normalization) and re-measure.
- **Cycle search limits:** Extremely dense graphs may hit the internal cap. We bound depth and the number of cycles gathered for determinism; note this in results if you approach those limits.

Minimal troubleshooting flow

1. **Re-run fresh:**
2. cd C:\Users\doug\aix\cthott
3. .\.venv\Scripts\Activate.ps1
4. python .\cthott_bench.py; python .\cthott_export.py
5. **Open artifacts:** start .\out\bench_report.html, ii .\out.
6. **If numbers differ:** run a **diff** on bench_report.json; inspect the changed metric's CSV (e.g., data\paraphrases.csv for PHI).
7. **If Yoneda coverage warnings:** python .\probe_sync.py --fill 0 → re-run bench → curate newly filled cells.
8. **Still stuck?** Share bench_report.json + the specific CSV; the issue is always localized to one of those.

With this layout and playbook, anyone can clone the folder, run the pipeline on a single PC, and verify your numbers and graphs—exactly the kind of **reproducible, auditable** workflow we want.

19. Appendices

A. Formal notes and proof sketches for each metric

A.1 Functor Consistency (Assoc/Id).
Let C\mathcal{C}C be a small category of expression types. For
f:X → Y,g:Y → Z,h:Z → Wf:X\!\to\!Y, g:Y\!\to\!Z, h:Z\!\to\!Wf:X→Y,g:Y→Z,h:Z→W,

h∘(g∘f)=(h∘g)∘f,idY∘f=f=f∘idX.h\circ(g\circ f)=(h\circ g)\circ f,\qquad \mathrm{id}_Y\circ f=f=f\circ \mathrm{id}_X.h∘(g∘f)=(h∘g)∘f,idY∘f=f=f∘idX.

Bench: interpret morphisms as total Python functions; evaluate both sides on a fixed payload. **Score** =1=1=1 iff equalities hold. **Soundness**: If functions are pure and composition is ordinary function composition, equalities hold extensionally.

A.2 Yoneda Probes & AUC.
Let P\mathcal{P}P be probes, vp:Σ→{0,1}v_p:\Sigma\to\{0,1\}vp:Σ→{0,1}. For a pair (a,b)(a,b)(a,b),

sim(a,b)=1|P|∑p∈P1{vp(a)=vp(b)}.\mathrm{sim}(a,b)=\frac1{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\mathbf{1}\{v_p(a)=v_p(b)\}.sim(a,b)=|P|1p∈P∑1{vp(a)=vp(b)}.

Report:

- **YonedaSynonym** =E[sim|label=1]=\mathbb{E}[\mathrm{sim}\mid \text{label}=1]=E[sim|label=1].
- **YonedaNonSyn** =1−E[sim|label=0]=1-\mathbb{E}[\mathrm{sim}\mid \text{label}=0]=1−E[sim|label=0].
- **YonedaAUC** = Mann–Whitney probability Pr[sim+>sim−]+12Pr[=]\Pr[\mathrm{sim}^+>\mathrm{sim}^-]+\tfrac12\Pr[=]Pr[sim+>sim−]+21Pr[=].
  Sketch: Values in [0,1][0,1][0,1] by construction; AUC is a rank-based probability in [0,1][0,1][0,1].

A.3 Pullback Coreference Score (PCS).
Given feature maps f:Cand→Ff:\mathrm{Cand}\to Ff:Cand→F, g:Pron→Fg:\mathrm{Pron}\to Fg:Pron→F, the **pullback**

Cand×FPron={(c,p)|f(c)=g(p)}\mathrm{Cand}\times_F\mathrm{Pron}=\{(c,p)\mid f(c)=g(p)\}Cand×FPron={(c,p)|f(c)=g(p)}

filters legal antecedents. Ranking ρ\rhoρ (subject≺object, recent≺older) selects c^\hat cc^. **PCS** = accuracy vs. gold. *Sketch*: Pullback enforces equalities; ranking is external and transparent.

A.4 Beck–Chevalley (BC) defect (normalized).
Let CCC be scores for cartesian-licensed path; NNN for noncartesian path.

BCDefectNorm=clamp[0,1](C¯−N¯+12).\mathrm{BCDefectNorm}=\mathrm{clamp}_{[0,1]}\left(\frac{\overline{C}-\overline{N}+1}{2}\right).BCDefectNorm=clamp[0,1](2C−N+1).

Sketch: Affine map sends C¯−N¯∈[−1,1]\overline{C}-\overline{N}\in[-1,1]C−N∈[−1,1] to [0,1][0,1][0,1].

A.5 Sheaf Gluing Ratio.
A "family" Ui↦siU_i\mapsto s_iUi↦si (local assignments) is **gluable** if on overlaps Ui∩UjU_i\cap U_jUi∩Uj the restrictions agree. Our toy unifier is string equality with a reserved conflict token. **Score** = #gluable/#families\#\text{gluable}/\#\text{families}#gluable/#families.

A.6 Transport Invariance.
For licensed paraphrase pairs (s1,s2)(s_1,s_2)(s1,s2) and evaluator EEE,

Transport=E[1{E(s1)=E(s2)}].\mathrm{Transport}=\mathbb{E}\big[\mathbf{1}\{E(s_1)=E(s_2)\}\big].Transport=E[1{E(s1)=E(s2)}].

Sketch: A naturality-style check; value in [0,1][0,1][0,1] is mean of indicator.

A.7 Paraphrase Homotopy Index.
In graph G=(V,E)G=(V,E)G=(V,E), let C\mathcal{C}C be simple cycles, C⋆={c∈C:|c|>2}\mathcal{C}_\star=\{c\in\mathcal{C}:|c|>2\}C⋆={c∈C:|c|>2}.

PHI=|C⋆||C|(legacy),PHI_nodes=|C⋆||V|.\mathrm{PHI}=\frac{|\mathcal{C}_\star|}{|\mathcal{C}|}\quad(\text{legacy}),\qquad \mathrm{PHI_nodes}=\frac{|\mathcal{C}_\star|}{|V|}.PHI=|C||C⋆|(legacy),PHI_nodes=|V||C⋆|.

Sketch: PHI penalizes 2-cycle domination; PHI_nodes is scale-free.

A.8 Univalence Witness.
Given e:A→Be:A\to Be:A→B, e−1:B→Ae^{-1}:B\to Ae−1:B→A with normalizer v\nuv,

v(e−1∘e(a))=v(a),e∘e−1(b)=b.\nu(e^{-1}\!\circ e(a))=\nu(a),\quad e\circ e^{-1}(b)=b.v(e−1∘e(a))=v(a),e∘e−1(b)=b.

Score =1=1=1 if both hold on a finite sample; else 0. *Sketch*: Checks bidirectional equivalence up to chosen normalization.

A.9 Curry–Uncurry Parity.
Isomorphism Hom(X×A,B)≅Hom(X,BA)\mathrm{Hom}(X\times A,B)\cong \mathrm{Hom}(X,B^A)Hom(X×A,B)≅Hom(X,BA). **Score** = proportion of (x,a)(x,a)(x,a) where f(x,a)=g(x)(a)f(x,a)=g(x)(a)f(x,a)=g(x)(a).

A.10 Presupposition Monad & Effect Leak.

Carrier

M P=(truth:B,effects:P(E))M\,P=(\mathrm{truth}:\mathbb{B},\mathrm{effects}:\mathcal{P}(E))MP=(truth:B,effects:P(E));

unit(v)=(v,∅)\mathrm{unit}(v)=(v,\emptyset)unit(v)=(v,∅), (v,E)\bindk=(v′,E∪E′)(v,E)\bindk=(v',E\cup E')(v,E)\bindk=(v′,E∪E′).

Leak check: compare main-clause truth with/without trigger; equal ⇒ 1, else 0.

B. CSV schemas (BNF), examples, and edge cases

Conventions: UTF-8, one header row, comma-separated, fields with commas/quotes **must** be quoted with inner quotes doubled.

B.1 Schemas (EBNF-like)

probe_truth.csv ::= header_pt newline { row_pt newline }

header_pt ::= "probe,word,value"

row_pt ::= probe "," word "," ("0"|"1") ["," meta*]

synonyms.csv ::= "w1,w2,label" newline { word "," word "," ("0"|"1") newline }

mentions.csv ::= "id,gender,num,role,recency" newline

{ id "," gender "," num "," role "," int newline }

pronouns.csv ::= "form,gender,num,gold" newline

{ form "," gender "," num "," id newline }

paraphrases.csv ::= "s1,s2,reason" newline

{ quoted "," quoted "," reason? newline }

discourse.csv ::= "family,key,value" newline

{ fam "," key "," value newline }

scope.csv ::= "cond,score" newline

{ ("cartesian"|"noncartesian") "," float01 newline }

transport.csv ::= "s1,s2" newline { quoted "," quoted newline }

Notes: quoted = " … " with "" escapes; float01 ∈ [0,1][0,1][0,1].

B.2 Examples (minimal)

probe_truth.csv

probe,word,value

finance?,bank,1

countable?,river,1

synonyms.csv

w1,w2,label

rock,anvil,1

bank,river,0

paraphrases.csv

s1,s2,reason

"The committee approved the plan.","The plan was approved by the committee.","active-passive"

B.3 Edge cases & guidance

- **Quoting**:
  ("He said, ""Go!""."","He told them to go.","report")
- **Duplicates**: avoid duplicate (probe,word); if present, last wins.
- **Coverage**: every w1,w2 should have values for **all** probes → run probe_sync.py.

- **Conflicts**: discourse.csv values starting with conflict: mark hard clashes.
- **Normalization**: Decide early on lowercasing/lemmatization; Yoneda compares exact strings.

C. Algorithmic pseudocode & complexity notes

C.1 Yoneda similarity & AUC

Input: probes P, truth T[p][w]∈{0,1}, pairs S={(a,b,label)}

```
for each (a,b,label) in S:
   sim = mean_p 1{T[p][a]==T[p][b]}
   if label==1: POS.append(sim) else NEG.append(sim)


# AUC by Mann–Whitney
wins=ties=0
for p in POS:
   for n in NEG:
      if p>n: wins++
      elif p==n: ties++
AUC = (wins + 0.5*ties) / (|POS|*|NEG|)
```

Time: O(|P||S| + |POS||NEG|).

C.2 Pullback coreference (PCS)

```
for each pronoun pr:
   C = [cand for cand in Mentions if f(cand)==g(pr)]
   sort C by (role=="subj" first, recency descending)
   pred = C[0].id if C else None
   acc += 1{pred == pr.gold}
PCS = acc / #pronouns
```

Time: O(#pronouns·#mentions).

C.3 BC-defect

C = mean(score where cond=="cartesian")

N = mean(score where cond=="noncartesian")

BC = clamp((C - N + 1)/2, 0, 1)

Time: O(#rows).

C.4 Gluing

ok=0; total=#families

for fam in families:

  conflict = ∃ key with disagreeing values OR value startswith "conflict:"

  if not conflict: ok++

Gluability = ok / total

Time: O(total·avg_keys).

C.5 Transport

matches = mean_{(s1,s2)} 1{E(s1)==E(s2)} # E = evaluator

Time: O(#pairs·eval_cost).

C.6 Paraphrase cycles (bounded DFS)

cycles = {}

for v in V:

  DFS(v, path=[], depth<=D, cap<=C)

  # record cycle when neighbor == start; dedupe by rotation-minimal tuple

PHI = |{c: len(c)>2}| / |cycles| # if |cycles|>0 else 0

PHI_nodes= |{c: len(c)>2}| / |V|

Time: bounded by depth D and cap C; practical ≈ O(V+E).

C.7 Univalence / Curry / Effect-Leak

Finite enumeration; O(1) to O(|X||A|) in our toy settings.

D. Command cheat sheet (PowerShell/Batch/Graphviz)

Setup & run

```
cd C:\Users\doug\aix\cthott
py -3 -m venv .venv
.\.venv\Scripts\Activate.ps1
python .\cthott_bench.py
python .\cthott_export.py
start .\out\bench_report.html
```

One-click

```
.\run_all.bat
```

Graphviz (optional)

```
winget install Graphviz.Graphviz -s winget
$env:Path += ";C:\Program Files\Graphviz\bin"
dot -Tpng .\out\paraphrase_graph.dot -o .\out\paraphrase_graph.png
ii .\out\paraphrase_graph.png
```

Probe coverage & rerun

```
python .\probe_sync.py --fill 0
python .\cthott_bench.py; python .\cthott_export.py; start .\out\bench_report.html
```

Diff reports

```
# (If you saved diff_report.py from §12)
python .\diff_report.py .\out\bench_report.json ..\prev\out\bench_report.json
```

Zip reproducibility bundle

```
pip freeze > .\out\env.txt
```

```
$stamp = (Get-Date).ToString("yyyyMMdd_HHmmss")
Compress-Archive -Force -Path `
  .\*.py, .\data\*, .\out\bench_report.json, .\out\env.txt `
  -DestinationPath ".\ai1-linguo_run_$stamp.zip"
```

Batch one-click content (already provided)

```
@echo off
cd /d %~dp0
if not exist .venv\Scripts\python.exe ( py -3 -m venv .venv )
call .venv\Scripts\activate.bat
python cthott_bench.py || goto :eof
python cthott_export.py || goto :eof
if exist "C:\Program Files\Graphviz\bin\dot.exe" (
  "C:\Program Files\Graphviz\bin\dot.exe" -Tpng out\paraphrase_graph.dot -o
out\paraphrase_graph.png
)
start "" out\bench_report.html
```

E. JSON report schema and exporter template

E.1 Minimal JSON schema (informal)

```
{
  "FunctorConsistency": 1.0,
  "YonedaSynonym": 1.0,
  "YonedaNonSyn": 0.667,
  "YonedaAUC": 1.0,
  "PCS": 1.0,
  "BCDefectNorm": 0.85,
```

```
  "Gluability": 0.667,
  "Transport": 1.0,
  "PHI": 0.25,
  "PHI_nodes": 0.286,
  "UA_witness": 1.0,
  "CurryParity": 1.0,
  "EffectLeakAudit": 1.0,

  "_meta": {
    "run_time": "2025-09-15T03:21:12Z",
    "python": "3.13.0",
    "bench_version": "git:abcdef1",
    "data_checksums": {
      "probe_truth.csv": "sha256:…",
      "synonyms.csv":    "sha256:…"
    },
    "host": "Windows-10-10.0.22631"
  }
}
```


Notes: _meta is optional but recommended (see §12). Keys are floats in [0,1][0,1][0,1].

E.2 Exporter HTML skeleton (what cthott_export.py writes)


```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>CT/HoTT-Linguo Bench Report</title>
```

```
<style>
body{font-family:Segoe UI,Tahoma,Arial,sans-serif;margin:24px}
table{border-collapse:collapse;margin-top:16px}
td,th{border:1px solid #ddd;padding:8px}
th{background:#f5f5f5}
.code{background:#f7f7f7;border:1px solid #eee;padding:8px;border-radius:6px;white-space:pre-wrap}
</style>
</head>
<body>
<h1>CT/HoTT-Linguo Bench Report <small><!-- timestamp --></small></h1>
<div>Data dir: <!-- absolute path --></div>
<table>
  <tr><th>Metric</th><th>Score (0..1)</th></tr>
  <!-- rows rendered in fixed order -->
</table>

<h3>Paraphrase graph (Graphviz DOT)</h3>
<p>Render to PNG if Graphviz is installed:</p>
<div class="code">dot -Tpng "…\paraphrase_graph.dot" -o "…\paraphrase_graph.png"</div>

<h3>Probe matrix CSV</h3>
<p>Saved to: <code>…\probe_matrix.csv</code></p>
</body>
</html>
```

E.3 Exporter responsibilities checklist

- Read out/bench_report.json; fail loudly if missing.
- Emit table in a fixed, human-friendly order.
- Write out/paraphrase_graph.dot from paraphrases.csv (duplicate edges to visualize undirected relationship).
- Write out/probe_matrix.csv by pivoting probe_truth.csv (words × probes).
- Avoid external dependencies; be deterministic (sort keys).

Closing note

These appendices give you the *nuts and bolts*—definitions, schemas, pseudocode, commands, and artifact formats—needed to audit, reproduce, and extend the bench. They're deliberately concise so you can map each concept to a specific file or row, make a change, and immediately see its effect in the JSON and graph.

References

Foundational Category Theory & HoTT

- Awodey, S. (2010). *Category Theory* (2nd ed.). Oxford University Press.
- Leinster, T. (2014). *Basic Category Theory*. Cambridge University Press.
- Mac Lane, S. (1998). *Categories for the Working Mathematician* (2nd ed.). Springer.
- Fong, B., & Spivak, D. I. (2019). *Seven Sketches in Compositionality: An Invitation to Applied Category Theory*. Cambridge University Press.
- Spivak, D. I. (2014). *Category Theory for the Sciences*. MIT Press.
- The Univalent Foundations Program. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study.

Categorical Logic, Beck–Chevalley, and Type Theory

- Jacobs, B. (1999). *Categorical Logic and Type Theory*. Elsevier.
- Johnstone, P. T. (2002). *Sketches of an Elephant: A Topos Theory Compendium*. Oxford University Press.
- Lambek, J., & Scott, P. J. (1986). *Introduction to Higher-Order Categorical Logic*. Cambridge University Press.

Semantics, Scope, and Pragmatics

- Barker, C., & Shan, C. (2014). *Continuations and Natural Language*. Oxford University Press.
- Heim, I., & Kratzer, A. (1998). *Semantics in Generative Grammar*. Blackwell.
- Jacobson, P. (2014). *Compositional Semantics: An Introduction to the Syntax/Semantics Interface*. Oxford University Press.
- Kamp, H., & Reyle, U. (1993). *From Discourse to Logic*. Kluwer.
- Montague, R. (1973). The proper treatment of quantification in ordinary English. In J. Hintikka et al. (Eds.), *Approaches to Natural Language* (pp. 221–242). Reidel.
- Beaver, D. I. (2001). *Presupposition and Assertion in Dynamic Semantics*. CSLI.

Category Theory in Linguistics / Compositional Distributional Semantics

- Coecke, B., Sadrzadeh, M., & Clark, S. (2010). Mathematical foundations for a compositional distributional model of meaning. *Linguistic Analysis*, 36, 345–384.
- Lambek, J. (1999). Type grammar revisited—pregroups and residues. *Logical Aspects of Computational Linguistics*, 95–109. (Also: *Journal of Logic and Algebraic Programming*, subsequent expositions.)

Monads, Effects, and Computational Semantics

- Moggi, E. (1991). Notions of computation and monads. *Information and Computation*, 93(1), 55–92.
- Wadler, P. (1992). The essence of functional programming. *POPL '92*, 1–14.
- Charlow, N. (2014). On the semantics of exceptional scope. *Natural Language Semantics*, 22(4), 353–397.
- Shan, C. (2005). Linguistic side effects. *COLING/ACL Tutorial*. (See also subsequent work on continuations and quantification.)

Coreference & Anaphora (feature and ranking heuristics)

- Hobbs, J. R. (1978). Resolving pronoun references. *Lingua*, 44(4), 311–338.
- Lappin, S., & Leass, H. J. (1994). An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4), 535–561.

Sheaves, Gluing, and Constraint Composition (contextual/logic background)

- Vickers, S. (1996). *Topology via Logic*. Cambridge University Press. (Background on sheaf/gluing intuitions.)
- Abramsky, S., & Brandenburger, A. (2011). The sheaf-theoretic structure of non-locality and contextuality. *New Journal of Physics*, 13, 113036. (Methodological sheaf background useful for constraint composition.)

Graphs, Evaluation, and Statistics

- Ellson, J., Gansner, E., Koutsofios, E., North, S., & Woodhull, G. (2002). Graphviz—Open source graph drawing tools. In *Graph Drawing* (pp. 483–484). Springer.
- Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a ROC curve. *Radiology*, 143(1), 29–36. (AUC fundamentals; equivalence to Mann–Whitney.)

Applied/Practical Texts (useful to practitioners)

- Milewski, B. (2018). *Category Theory for Programmers*. (Self-published; widely used in practice.)
- Rijke, E. (2022). *Introduction to Homotopy Type Theory*. Cambridge University Press.

Prior & Companion Work by the Author

- Youvan, D. C. (2025). *Language Runs on CT/HoTT: Compositional Semantics, Homotopical Identity, and a Testable Compliance Battery*. ResearchGate. DOI: **10.13140/RG.2.2.27725.91364**.
- Youvan, D. C. (2025). *Achieving Artificial Intelligence (youvan.ai1) on a PC with Category Theory & HoTT*. ResearchGate. DOI: **10.13140/RG.2.2.36085.03047**.

Notes on usage

- Citations above are chosen to ground each metric or engineering choice: categorical laws (Mac Lane; Awodey), Yoneda/CT background (Fong & Spivak; Leinster), HoTT/univalence (UF book), scope/continuations (Barker & Shan), presupposition (Beaver), monads (Moggi; Wadler), coreference heuristics (Hobbs; Lappin & Leass), and graph/ROC fundamentals (Graphviz; Hanley & McNeil).
- Where a specific construction appears across multiple sources (e.g., Beck–Chevalley), Jacobs (1999) and Johnstone (2002) provide standard treatments.

FOR IMMEDIATE RELEASE

youvan.ai Unveils ai1 and ai1-Linguo: Local-First, Executable AI Grounded in Category Theory & HoTT

*Two complementary releases show how to **build** and **verify** AI on a single PC—turning abstract mathematics into reproducible engineering.*

FRONTENAC, Kansas — September 15, 2025 — **youvan.ai** today announced two coordinated research releases that demonstrate a local-first approach to artificial intelligence built on **Category Theory (CT)** and **Homotopy Type Theory (HoTT)**:

1. **Achieving Artificial Intelligence (youvan.ai1) on a PC with Category Theory & HoTT** — a compact, deterministic **equational engine** that performs typed term rewriting to normal form with machine-checkable witnesses.
   Published: September 2025 • **DOI:** 10.13140/RG.2.2.36085.03047
   Author: Douglas C. Youvan

2. **ai1-Linguo: A CT/HoTT Compliance Bench for Language on a Single PC** — an **executable benchmark** that tests whether linguistic operations respect CT/HoTT structure (functoriality, pullbacks, Beck–Chevalley scope, sheaf gluing, homotopies, univalence, curry/uncurry, presupposition effects).
   Publishing today on ResearchGate • DOI: *to be inserted by the author upon assignment*
   Author: Douglas C. Youvan

“Most systems minimize loss; **ai1** and **ai1-Linguo** verify **laws**,” said **Douglas C. Youvan**, founder of youvan.ai. “The engine builds compositional explanations and witnesses; the bench audits whether transformations preserve the equations they should. Both run on a single PC and emit human-readable artifacts.”

What’s inside (copy-safe math, in-line)

- **ai1 (engine):** typed term rewriting with identities and composition enforced by construction, e.g.
  compose(id(Y), f) == f == compose(f, id(X)) and
  compose(h, compose(g, f)) == compose(compose(h, g), f)
  Supports mid-identity elimination compose(g, id(B), f) -> compose(g, f), η-style harmless expansions, and pushout β-laws via small horn patterns. Every run emits a **JSON trace** of equalities used.

- **ai1-Linguo (bench):** CSV-in → **compliance battery** (0..1) → JSON/HTML + Graphviz DOT/PNG. Metrics include:
  • **FunctorConsistency** (assoc/identity as above)

- **Yoneda** probes for meaning-as-use (**Synonym**, **NonSyn**, **AUC**)
- **PCS** (pullback coreference via feature agreement)
- **BCDefectNorm** (scope commutation fitness)
- **Gluability** (sheaf-style discourse compatibility)
- **Transport** (licensed substitutions: evaluator E(s1)==E(s2))
- **PHI / PHI_nodes** (nontrivial paraphrase cycles per node)
- **UA_witness** (univalence by round-trip witnesses: e: A->B, e_inv: B->A, with e_inv(e(a))==a and e(e_inv(b))==b)
- **CurryParity** (Hom(X*A, B) ~= Hom(X, B^A) checked by enumeration)
- **EffectLeakAudit** (presupposition monad: bind((v,E), k) -> (v', union(E,E')); removing a trigger must keep main-clause truth unchanged)

Why it matters

Researchers, educators, and builders can now **inspect and reproduce** structural claims about language behavior **offline** without specialized infrastructure. The approach favors **interpretability**, **auditability**, and **determinism**: identical inputs yield identical JSON scores and DOT/PNG graphs, suitable for CI gates, dataset curation, and pedagogy.

Availability

- **Paper 1:** *Achieving Artificial Intelligence (youvan.ai1) on a PC with Category Theory & HoTT* — ResearchGate, DOI **10.13140/RG.2.2.36085.03047**
- **Paper 2:** *ai1-Linguo: A CT/HoTT Compliance Bench for Language on a Single PC* — ResearchGate, **DOI: [to be inserted]**
- **Company: youvan.ai** — https://www.youvan.ai
- **Contact: Douglas C. Youvan** — **doug@youvan.com**

About youvan.ai

youvan.ai develops **local-first, interpretable AI** grounded in Category Theory and HoTT. Our tools emphasize deterministic rewriting, compositional explanations, and machine-checkable witnesses—so teams can audit, reproduce, and trust what their systems are doing.

Media Contact
Douglas C. Youvan
youvan.ai • Frontenac, Kansas, USA
doug@youvan.com