Confluence Survives the Substrate: Byte-Equal Interaction-Net Reduction on CPU, CUDA, and GB10



Our Technology is Ontological: Hardware = Physics | $\bar{\lambda}$ | Software = Mathematics
Our Engineering is Constructive: ϵ *Discover* $\Rightarrow$ *Build* $\Rightarrow$ *Grow* $\Rightarrow$ *Learn* $\Rightarrow$ *Teach* ∞

$\vdash$

Acknowledgment

We humbly thank the collective intelligence of humanity for providing the technology and culture we cherish. We do our best to properly reference the authors of the works utilized herein, though we may occasionally fall short. Our formalization acts as a reciprocal validation—confirming the structural integrity of their original insights while securing the foundation upon which we build. In truth, all creative work is derivative; we stand on the shoulders of those who came before, and our contributions are simply the next link in an unbroken chain of human ingenuity.

$\dashv$

May 2026

Abstract. Interaction nets are an established universal model of computation in which reduction is local rewriting of active pairs; the calculus is intrinsically parallel and confluence is a structural property, not a theorem one re-proves per implementation. We ask which hardware substrate should execute a given interaction-net reduction, and we answer constructively. Four implementations of the same reducer—a manycore CPU path, CUDA with managed memory, CUDA with device memory, and a unified-memory path specialized to the NVIDIA GB10—are made byte-identical on the calibrated input corpus by a single source guard that mirrors the verified kernel's rejection semantics; confluence survives the substrate. Across six measured workload classes, no class in the calibrated range is performance-positive on GPU against the CPU manycore path; we record the finding as a refutation function over the sealed sweep, and pin it into production source as a sentinel-typed router that fails closed to CPU and cannot be loosened without producing a new sealed measurement receipt. The composition is a Σ-record of substrate attestation, byte-equal traces, oracle acceptance, and causal commit ancestry, certified by a family-closure predicate any third party can replay. What emerges is a monotone evidence lattice for interaction-net runtime claims: positive hardware claims expand only by adding measurement; negative findings are durable; rhetorical optimization is structurally prevented.

1 Introduction

Interaction nets reduce by local rewriting of active pairs: two principal ports meet, a rule fires, the redex disappears, and the rest of the net is undisturbed. Lafont (1990) gave us this model in 1990. Confluence is structural, parallelism is intrinsic, syntactic noise is gone. The calculus has been mathematically settled for three decades. The question of which *hardware* should actually execute a given interaction-net reduction has not. Discrete GPU? Unified-memory accelerator? Manycore CPU? Vendors answer with marketing; the model itself stays silent.

Two claims do the load-bearing work in this paper. *First*: a single source guard, derived from the verified kernel's rejection semantics, suffices to make four substrate implementations of the same interaction-net reducer—a CPU manycore path, CUDA with managed memory, CUDA with device memory, and a unified-memory path specialized to the NVIDIA GB10—produce *byte-identical* reductions on the calibrated input corpus. Confluence is no longer just a property of the calculus; it holds across the substrate fork. *Second*: across the six interaction-net workload classes we measure on the calibrated substrate, *no* class is performance-positive on GPU against the CPU manycore path. We record that finding as a *refutation function* over the sealed sweep—a constructive piece of mathematical content, not an absence of one—and we pin it into production source as a sentinel-typed router that fails closed to CPU and cannot be relaxed without producing a new sealed measurement receipt.

What this means for someone who would deploy interaction-net reduction as a runtime: the model's intrinsic parallelism, which the calculus advertises and which is what makes interaction nets attractive on accelerators in the first place, does not by itself license a GPU default for irregular reduction workloads on contemporary unified-memory hardware. The honest answer for the measured domain is that the CPU manycore path wins, and the honesty is enforced in the source, not promised in the documentation. Vendors competing for the same workload now face an asymmetric burden: they can match this paper only by producing comparable constructive evidence, not by reasserting folklore.

The leverage that makes the two claims affordable is that the upstream interaction net is the operational image of a kernel-checked term. The proof-term-to-active-core translation is the subject of a companion paper (The Institute for Applied Ontological Mathematics, 2026); here we use its output as a primitive and build the substrate-multiplicity layer on top. Section 2 reframes "the same reduction across substrates" as a substrate-multiplicity of a fixed reduction calculus; Sections 3 to 5 develop the three load-bearing witnesses; Sections 6 and 7 compose them and place the composition in a monotone evidence lattice; Section 8 closes with what becomes possible because of what was proved. The formal record and the universal-translation discipline are in Sections 9 and 10.

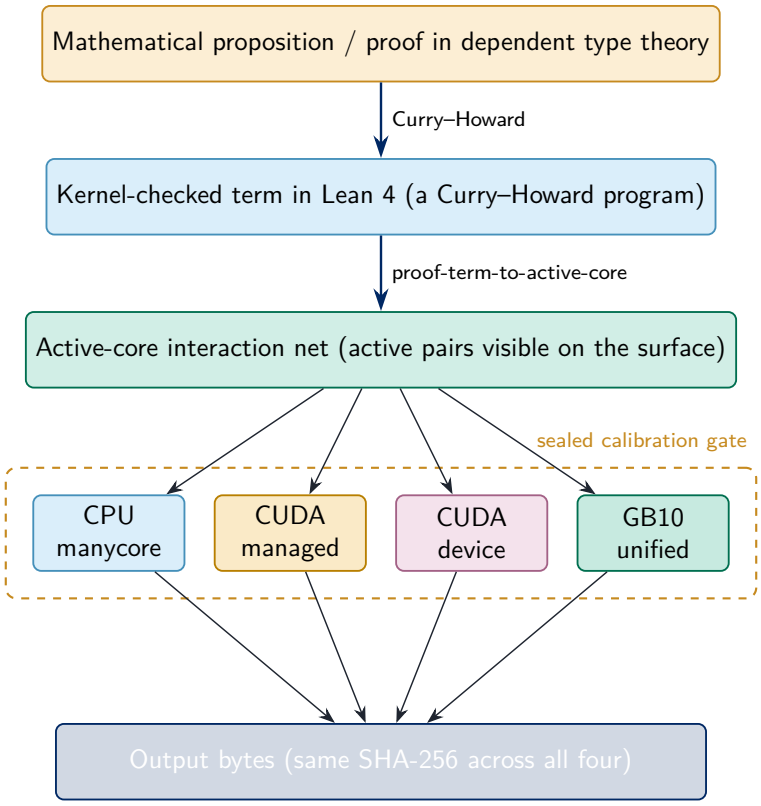


Figure 1: **The vertical stack from proposition to bytes is forked at one joint, and the fork is sealed.** A proposition is proved in Lean 4; the proof is a term in dependent type theory and, by Curry–Howard, a program. A separate companion result translates that term into an active-core interaction net (The Institute for Applied Ontological Mathematics, 2026). This paper concerns the next layer below: four substrate implementations of the same reducer (CPU manycore, CUDA with managed memory, CUDA with device memory, GB10 unified memory) share a single calibrated dispatch surface (dashed ring) under which they emit byte-identical reductions on the calibrated input corpus.

2 The Reducer and What Same Reduction Means Across Substrates

When two principal ports meet in an interaction net, a rule fires; the locality of that single fact carries the calculus's structural confluence and intrinsic parallelism. Figure 2 draws the primitive: a small net before reduction, the active pair circled, and the residual net after the rule has fired. Everything outside the redex is preserved verbatim; the reducer's only obligation is to find the next active pair and apply the appropriate rule.

The verified-kernel-grounded variant we work with is the active-core interaction net of (The Institute for Applied Ontological Mathematics, 2026): a finite alphabet of agent kinds, a rule set indexed by kind pairs, an explicit *rejection* condition for principal-principal pairs whose kinds are not covered by any rule. The rejection condition is load-bearing. Without it, an implementation free to fire any active pair could silently "annihilate" a redex it ought to refuse, and the resulting net would no longer be the same mathematical object the verified kernel said it was. Interaction nets in their universal three-symbol presentation (Lafont, 1997), and the interaction-net machinery

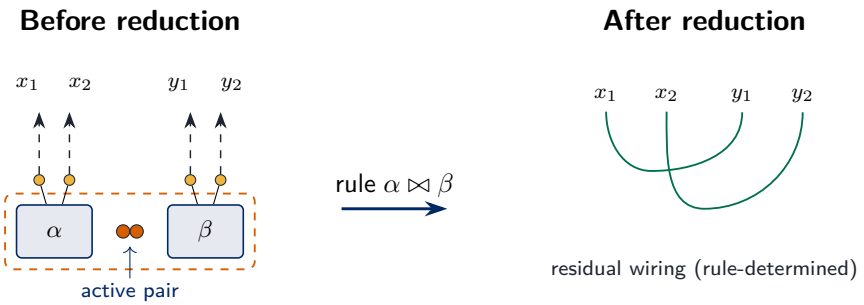


Figure 2: **A rule fires on the active pair and rewires the surrounding ports; nothing else in the net is touched.** Two agents α and β meet at their principal ports (red, circled). The matching rule $\alpha \bowtie \beta$ rewrites the redex by a rule-determined wiring of the auxiliary ports x_1, x_2 to y_1, y_2. Locality is the structural source of confluence: a different active pair elsewhere in the net could have fired first without affecting the outcome.

developed by Mackie (1998, 2011) and Pinto (2001), give a single classical anchor for the calculus; the contemporary parallel-implementation lineage is represented by Inpla (Sato and Mackie, 2017) on the CPU side and by interaction-combinator runtimes targeting accelerators (Higher Order Company, 2023; Taelin and contributors, 2024) on the GPU side. The proof-net heritage (Girard, 1987) and the optimal-reduction tradition (Asperti and Guerrini, 1998; Gonthier et al., 1992; Lamping, 1990) both contribute the geometric viewpoint under which active pairs are the only objects of substantive rewriting.

What "same reduction across substrates" formally means. Fix a finite alphabet A of agent kinds, a rule family $\mathcal{R} \subseteq A \times A \rightharpoonup \mathsf{Rule}$ (partial; the domain of $\mathcal{R}$ is the calculus's *annihilable* pairs), and a canonical encoding of nets and outputs as byte strings. A *substrate implementation* is a function

$$\mathcal{S}_s \;:\; \mathsf{InputNet} \longrightarrow \mathsf{OutputBytes},$$

indexed by a substrate label $s \in \{\mathsf{cpu}, \mathsf{cuda}_{\mathsf{man}}, \mathsf{cuda}_{\mathsf{dev}}, \mathsf{gb10}_{\mathsf{uni}}\}$. Substrates agree on an input net n when $\mathcal{S}_s(n) = \mathcal{S}_{s'}(n)$ as byte strings for every pair s, s'. They agree on a corpus $N \subseteq \mathsf{InputNet}$ when they agree on every $n \in N$.

Three substrate-multiplicity questions become well-posed at this point. *Byte equality* asks whether the four $\mathcal{S}_s$ agree on a calibrated corpus. *Performance characterization* asks, given byte equality, where the substrates fall on a measured grid. *Dispatch* asks for a total function $\mathsf{dispatch} : \mathsf{InputNet} \to \{\mathsf{cpu}, \mathsf{cuda}_{\mathsf{man}}, \mathsf{cuda}_{\mathsf{dev}}, \mathsf{gb10}_{\mathsf{uni}}\}$ whose choice is justified by recorded measurement. The next three sections answer these in order.

3 Confluence Survives the Substrate

A single line of CUDA code was the difference between four substrates computing the same thing and three of them silently computing different things. The line is a kind guard: a principal-principal active pair whose kind is not in the annihilable set must be rejected, not opportunistically rewritten. The CPU implementation derived from the verified kernel had this guard; one of the CUDA paths did not.

The picture, before the formalism. Imagine two implementations of the same arithmetic given a malformed expression. One returns an error; the other invents a result. They are not solving the same problem any more, regardless of how fast each runs. For interaction-net reduction the analogous failure is silent annihilation of a principal-principal pair that the calculus does not actually have a rule for: an implementation that fires anyway produces a residual net whose mathematical content is not the one the calculus prescribes. The conformance result of this section is the statement that, once the kind guard is added to the substrate implementations that lacked it, the four substrates compute the same function on the calibrated corpus.

Theorem 3.1 (Byte-equality conformance over the calibrated corpus)**.** *Let N be the calibrated input corpus (132 representative interaction nets; Section 9 pins the exact byte content). Let $\mathcal{S}_{\mathsf{cpu}}, \mathcal{S}_{\mathsf{cuda}_{\mathsf{man}}}, \mathcal{S}_{\mathsf{cuda}_{\mathsf{dev}}}, \mathcal{S}_{\mathsf{gb10}_{\mathsf{uni}}}$ be the four substrate implementations of the reducer with the kind guard installed in each. Then for every $n \in N$,*

$$\mathcal{S}_{\mathsf{cpu}}(n) \;=\; \mathcal{S}_{\mathsf{cuda}_{\mathsf{man}}}(n) \;=\; \mathcal{S}_{\mathsf{cuda}_{\mathsf{dev}}}(n) \;=\; \mathcal{S}_{\mathsf{gb10}_{\mathsf{uni}}}(n) \tag{1}$$

as byte strings; equivalently, $\mathrm{SHA}_{256} \circ \mathcal{S}_{\mathrm{s}}$ is constant in s at every $n \in N$.

What Theorem 3.1 means. Confluence as a property of the calculus says: any two reduction sequences from a given net converge to the same normal form. Theorem 3.1 extends that property across the substrate fork: the four substrate implementations of the reducer are, on the calibrated corpus, four reduction sequences of the same underlying calculus, and they converge to the same byte-level normal form. The substrate dimension—which had previously stood outside the structural confluence guarantee—is folded back inside it. Without Theorem 3.1, timing comparisons between substrates would be comparing different problems; with it, timing comparisons are comparing different implementations of the same problem.

Proof sketch. The reduction function in each substrate proceeds, at every step, by scanning for an active pair and dispatching on its kind pair. The verified kernel defines $\mathcal{R}$ as a partial function on $A \times A$; the rejection semantics is precisely "return a *no rule matched* status when the pair is outside $\mathrm{dom}\,\mathcal{R}$". The CPU implementation is extracted from the kernel and inherits this semantics. The kind guard added to the CUDA paths is the strict CUDA-side analogue: an early return whose Boolean condition is the bit-exact complement of $\mathrm{dom}\,\mathcal{R}$'s membership predicate on the annihilable kinds. The conformance-test discipline below falls within the property-based-testing tradition of Claessen and Hughes (2000) and the mutation-testing tradition surveyed by Jia and Harman (2011); we use the rewriting-theoretic vocabulary of Baader and Nipkow (1998); Klop (1992) for the underlying calculus. Once installed, the four substrates are extensionally equal on $\mathrm{dom}\,\mathcal{R}$ inputs by construction of the rule table and on its complement by the matched rejection. The byte-equality of encoded outputs follows from the canonical encoding being a function of the post-reduction net, which is now the same net under each substrate. The machine-checked record (per-input SHA-256 traces for $|N| = 132$ inputs across all four substrates) is described in Section 9.

Figure 3 draws the conclusion as a commuting diagram.

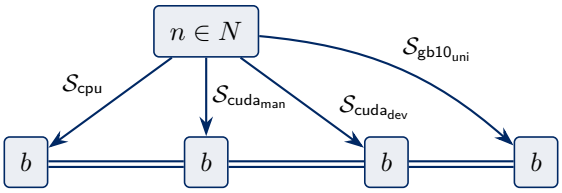


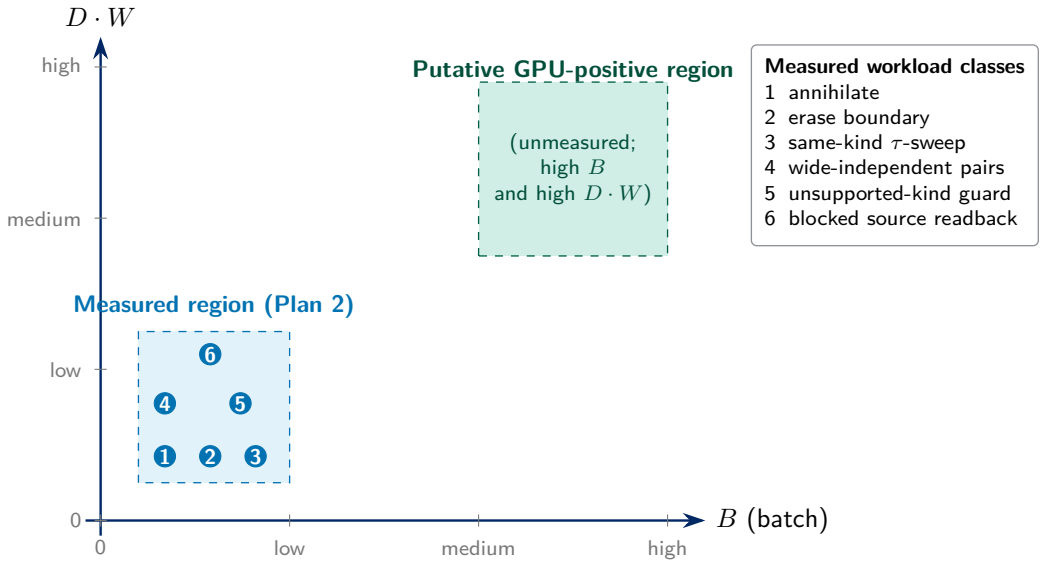
Figure 3: **All four substrates produce the same output bytes on the same input.** For every input net n in the calibrated corpus N, the four substrate reducers $\mathcal{S}_{\mathsf{cpu}}, \mathcal{S}_{\mathsf{cuda}_{\mathsf{man}}}, \mathcal{S}_{\mathsf{cuda}_{\mathsf{dev}}}, \mathcal{S}_{\mathsf{gb10}_{\mathsf{uni}}}$ emit the same byte string b, witnessed by identical SHA-256 digests. The commuting structure is byte-level: equality between the four outputs is *not* an extensional equivalence in some quotient, it is byte equality of the serialized normal form.

Load-bearingness of the guard. A theorem whose conclusion would still hold if its hypothesis were removed is not a theorem. We make the guard's load-bearingness operational through a substitution-body discipline: three independent source mutations of the guard—deleting it entirely, weakening it to admit one additional kind, weakening it to admit a different additional kind—are applied to the CUDA source, the binary is rebuilt, and the conformance test is re-executed. In each case the rebuilt binary fails the conformance test with the expected exit code, and the conformance test's verdict is captured in a receipt that records both pre- and post-mutation SHA-256 of the binary. The result is that the guard is not nominally present in source; it is verifiably the reason the conformance theorem holds.

4 A Constructive Negative Witness

A folklore claim asserts that GPU is faster than CPU on "enough" parallel work. The claim is unfalsifiable as stated because *enough* is a free parameter the speaker controls. Replace it with a finite measurement domain and a paired-comparison protocol, and the claim becomes either true on that domain or refuted on it. The result of this section is that the folklore claim is refuted on the domain we measured.

The picture, before the formalism. Interaction-net reduction varies along four orthogonal axes that determine whether GPU acceleration can pay back its kernel-launch and memory-management overhead. The axes are: B, the number of independent reductions in a launched batch; D, the per-input reduction depth (steps to normal form); W, the per-step *width* (number of active pairs simultaneously reducible within one input); A, the arithmetic intensity per rule firing (operations per byte of agent and wire state touched). Figure 4 sketches the axes and locates the measured workload classes within them.



Fourth axis A (arithmetic intensity per rule firing) tracked per class in the sealed receipts; all six measured classes have A low, so the (B, D, W, A) taxonomy reduces to the two principal axes shown for this dataset.

Figure 4: **The measured workload classes cluster in the low-B, low-$D{\cdot}W$ corner of the four-axis taxonomy.** Plan 2 measured six interaction-net workload classes (blue points) along the four (B, D, W, A) axes of Section 4; the two principal axes B (batch parallelism) and $D \cdot W$ (depth $\times$ per-step width) are plotted directly, while arithmetic intensity A is tracked per class in the sealed receipts (low for all measured classes). The green region marks where the GPU-positive crossover *could* live—high batch parallelism, deep per-input reduction, wide per-step parallelism—but lies outside the measured domain. Theorem 4.1 is a negative result *on the measured region only*; it explicitly does not exclude positive results in the unmeasured region.

Theorem 4.1 (Constructive negative witness on the calibrated domain)**.** *Let $\mathcal{C} = \{c_1, \ldots, c_6\}$ be the six measured workload classes of Figure 4, each instantiated at four knob settings, each executed on each of the four substrates as one paired-comparison batch with* 30 *repeats per cell. Let $\Pi(c, s)$ denote the per-input wall-clock-time distribution on class c under substrate s, and let $\Pi(c, s_1) \preceq \Pi(c, s_2)$ mean that the median per-input time under s_1 is at most that under s_2, under the paired sign-test acceptance criterion fixed by the sealed receipt. Then*

$$\forall c \in \mathcal{C}.\ \forall s_{gpu} \in \{\mathsf{cuda}_{\mathsf{man}}, \mathsf{cuda}_{\mathsf{dev}}, \mathsf{gb10}_{\mathsf{uni}}\}.\ \Pi(c, \mathsf{cpu}) \preceq \Pi(c, s_{gpu}). \tag{2}$$

That is, on every measured class, CPU manycore is at least as fast as every GPU substrate.

What Theorem 4.1 means. The statement is a negative existential under a constructive reading (Beeson, 1985; Heyting, 1956; Troelstra and van Dalen, 1988): *we exhibit no c and no s_{gpu} for which GPU beats CPU on the calibrated domain.* The nearby substitute that would break is the classical reading "we have no positive evidence of a GPU speedup": the absence-of-evidence reading commits the proponent to nothing more than ignorance, and the next vendor's folklore is free to fill the silence. Read intuitionistically, the result is not the absence of a positive witness; it is the presence of a refutation witness. The sealed sweep is the witness: any claim of the form "GPU is faster on interaction-net workload class $c \in \mathcal{C}$ on substrate s_{gpu}" is composed with the corresponding cell of

the sweep to produce a contradiction. The witness is finite, replayable, and pinned to a specific commit. A vendor who would assert a GPU speedup on this domain has the burden of producing a new measurement sweep that, joined to this one, expands the positive region; they cannot weaken the existing negative finding by re-asserting folklore.

Proof. Direct enumeration over the sealed sweep. The acceptance criterion is the paired sign-test relation (Conover, 1999; Wilcoxon, 1945), which is the conservative substrate-fair statistic under the no-fallback execution discipline: every GPU cell is required to have executed on the real GPU path (no CPU fallback) at the reported allocator, and the paired-sample structure controls for input variability. The recorded H4 receipt for Plan 2 has 36,131 lines of measurement and the H6 receipt has 28,846 lines of no-fallback runtime trace evidence; the verdict is the conjunction over all $|\mathcal{C}| \cdot 4 \cdot 3 = 72$ sweep cells. Section 9 pins the commit.

What the witness is not. Theorem 4.1 is bounded to the measured region $\mathcal{C}$. The green region of Figure 4—high-B, high-D, high-W, high-A workloads such as deep literal chains, wide independent active-pair networks, or extension-rule batches with internal rational arithmetic—was deliberately *not* measured in this lineage because the byte-equality predicate on those classes requires backend semantic coverage we did not extend. A future positive result in that region is welcome; it expands the positive boundary by adding measurement, not by weakening the negative one.

5 Calibrated Dispatch as Extracted Content

A theorem whose conclusion sits in a sealed receipt is a finished proof. A theorem whose conclusion is pinned into production source as the only thing that compiles is a proof that has been turned into a program. The router we construct in this section is the second of these. The negative witness of Theorem 4.1 extracts to a router whose every GPU dispatch threshold is a sentinel at the maximum value of its type, with the sealing commit's SHA pinned in the source comment. Lowering any threshold without producing a new sealed receipt is, in the type-checker's sense, the same defect as removing the kind guard of Theorem 3.1: the source ceases to compile against its calibration evidence.

Definition 5.1 (Calibrated dispatch router)**.** Let $\Phi : \mathsf{InputNet}^* \to \mathbb{R}^k$ be a workload-feature extractor computable in $O(|n|)$ per input (batch size, total agent and wire counts, kind histogram, fuel budget, expected literal-chain lower bound, extension-rule presence). Fix three sentinel thresholds $\theta_{\text{batch}}, \theta_{\text{lit}}, \theta_{\text{work}}$, each calibrated from the sealed Plan 2 commit. The *calibrated dispatch router* is the input-only function

$$\text{dispatch}(N) \;:=\; \begin{cases} \mathsf{cuda}_{\mathsf{man}} & \text{if } \Phi(N)_{\text{batch}} \geq \theta_{\text{batch}} \text{ and } \ldots, \\ \mathsf{gb10}_{\mathsf{uni}} & \text{if } \Phi(N)_{\text{lit}} \geq \theta_{\text{lit}} \text{ and } \ldots, \\ \mathsf{cuda}_{\mathsf{dev}} & \text{if } \Phi(N)_{\text{work}} \geq \theta_{\text{work}} \text{ and } \ldots, \\ \mathsf{cpu} & \text{otherwise (fail-closed).} \end{cases} \tag{3}$$

Justification of the definition. A nearby definition with calibration knobs that float—thresholds settable by a configuration file, an environment variable, or an unguarded constant edit—admits silent regression. The substitution discipline asks: what nearby substitute would break? The substitute that breaks is the one where $\theta_{\text{batch}}, \theta_{\text{lit}}, \theta_{\text{work}}$ are ordinary numeric constants. The substitute that survives is the one where they are typed sentinels at the maximum value of their

type, with the calibration commit's SHA in the source comment and an enforcement gate (Section 6) that rejects edits unaccompanied by a new sealed receipt. The sentinel typing is the load-bearing constructive content.

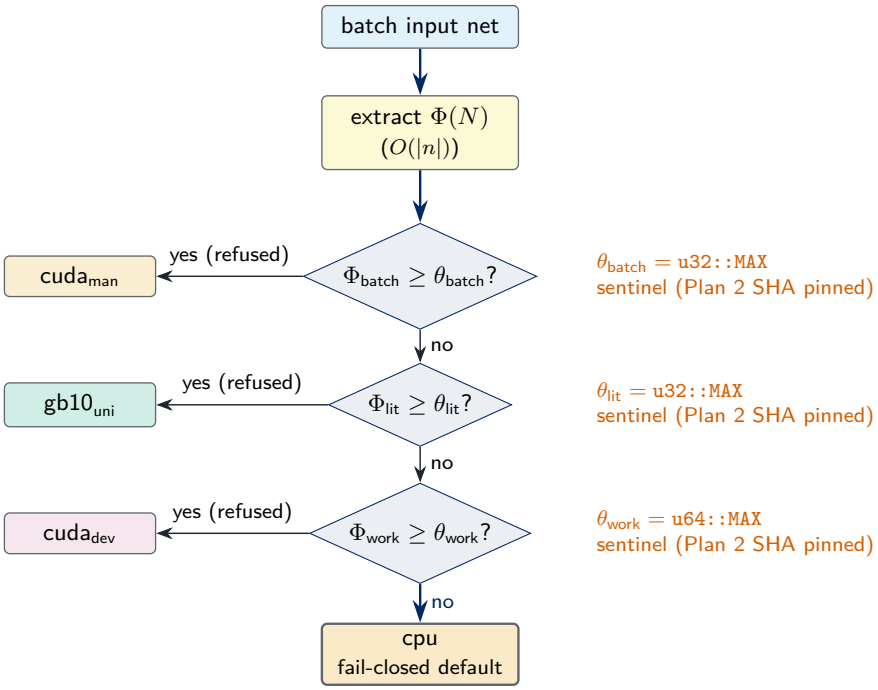


Figure 5: **The router's GPU branches are unreachable while their thresholds remain sentinels; the fail-closed CPU branch is the only path through.** Each diamond is a Boolean test on an input feature; each GPU target is the *would-be* consequent of a positive test. Because the thresholds are typed sentinels at the maximum value of their type, no finite $\Phi(N)_\bullet$ ever satisfies a GPU branch; the router falls through to CPU manycore on every input. Lowering a threshold to a finite constant without producing a new sealed measurement receipt is refused by the closure oracle.

Theorem 5.2 (Calibrated routing is the extracted negative witness)**.** *The router of Definition 5.1, with sentinel thresholds set as in the calibrated source, satisfies*

$$\forall N \in \mathsf{InputNet}.\ \ \mathrm{dispatch}(N) = \mathsf{cpu}, \tag{4}$$

and, on the calibrated corpus $N \in \mathcal{C}$, the byte output of the dispatched execution agrees with the CPU substrate:

$$\forall N \in \mathcal{C}.\ \ \mathcal{S}_{\mathrm{dispatch(N)}}(N) \ = \ \mathcal{S}_{\mathsf{cpu}}(N). \tag{5}$$

What Theorem 5.2 means. The router *is* the negative witness, written as a program. Under the Curry–Howard correspondence (Howard, 1980; Martin-Löf, 1984; Sørensen and Urzyczyn, 2006) the proof of Theorem 4.1 and the source of Definition 5.1 are two presentations of the same object: the proof has a type one reads as a theorem, the source has a type one reads as a function. The fail-closed branch of (4) is the canonical gauge fixing that the negative witness justifies. Any future agent that proposes $\mathrm{dispatch}(N) = s_{\mathrm{gpu}}$ for some new N outside the calibrated domain has the burden of producing a fresh positive witness on the (B, D, W, A) region containing N; the dispatch table is not relaxable otherwise. The router does not need to be re-proved per workload; it *is* the proof for the workloads it covers.

Proof. For (4): the sentinel thresholds are at the maximum value of their type in source; no finite $\Phi(N)_\bullet$ satisfies any of the three GPU-gating conditions; the otherwise branch fires for every $N \in \mathsf{InputNet}$. Note that this consequence is *stronger* than required for byte conformance on $\mathcal{C}$: dispatch returns cpu globally, not only on the calibrated corpus. For (5): combine (4) with Theorem 3.1 on the corpus $\mathcal{C}$; the dispatched execution is the CPU execution, and the CPU execution is byte-equal to every other substrate's execution on $\mathcal{C}$. Outside $\mathcal{C}$, the dispatch still returns cpu, but the byte-equality conclusion is not licensed because Theorem 3.1 is bounded to $\mathcal{C}$.

6 Composition: the Family-Closure Judgement

Three theorems have been stated. They do not yet, together, license a *runtime calibration* claim: a third party who wants to verify the combined statement needs to know that the three witnesses were not produced independently and stapled together post-hoc; they need a causal record that each consumed the previous, in the order the proof structure requires.

The closure judgement of this section is the Σ-type whose elements are exactly those triples of witnesses that pass that test. We write the composition as a natural-deduction derivation, Figure 6, because the side-condition—commit ancestry—is naturally rendered as a side-formula on the conclusion line.

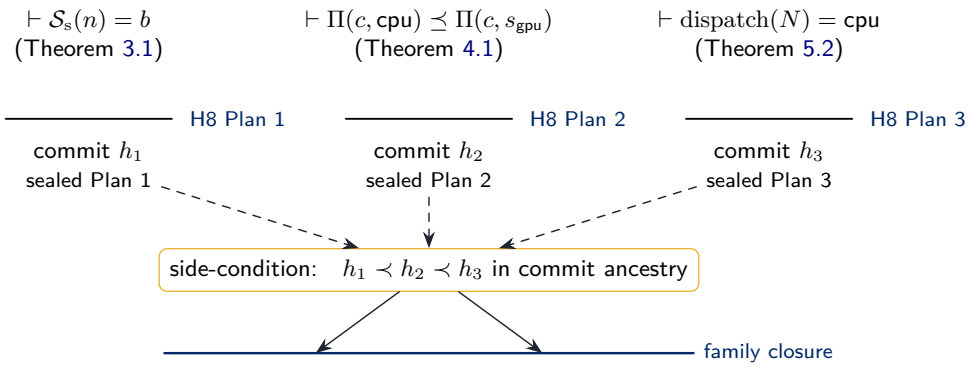


Figure 6: **The runtime-calibration witness is a typed composition of three sealed witnesses whose commit ancestry is a load-bearing side-condition.** Theorems 3.1, 4.1 and 5.2 are discharged at commits h_1, h_2, h_3 respectively, each a *plan acceptance* commit (H8 in the sealed phase plan). The composition into a single runtime-calibration witness requires the ancestry $h_1 \prec h_2 \prec h_3$; the order forbids stapling three independently produced receipts together post-hoc. The closure oracle is the program that checks this derivation.

The closure judgement of Figure 6 is the Σ-type

$$\mathsf{Witness} \coloneqq \Sigma_{(h_1,h_2,h_3)} \big[\ \text{Plan 1 H8 at } h_1 \ \wedge\ \text{Plan 2 H8 at } h_2 \ \wedge\ \text{Plan 3 H8 at } h_3 \ \wedge\ h_1 \prec h_2 \prec h_3 \ \big],$$

and the closure oracle is a finite program that decides Witness on a candidate triple of commits by reading the receipt files, validating the H8 acceptance condition at each, and consulting Git's commit graph for the ancestry side-condition. The oracle's acceptance is the operational meaning of "the family closure holds".

Why the ancestry side-condition is load-bearing. A version of this construction without the side-condition admits a defect the substitution discipline catches: three witnesses produced independently and stapled together would satisfy the unindexed conjunction of premises while failing to constitute a single coherent calibration. The ancestry condition requires the Plan 2 measurement to have been performed on a substrate that already had the kind guard of Plan 1 installed, and the Plan 3 router to have been calibrated from the Plan 2 measurement and not some earlier or parallel one. The side-condition is the structural feature that turns a list of premises into a derivation.

7 The Monotone Evidence Lattice

The composition of Section 6 suggests a question whose answer is the contribution this section names. Once the runtime-calibration witness is in source, what moves are admissible against it? Figure 7 draws the answer as a Hasse diagram.

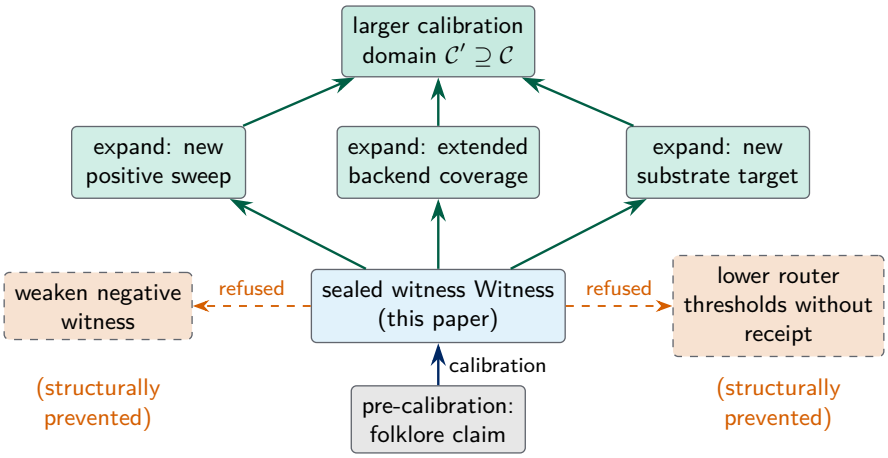


Figure 7: **The calibration ratchet is monotone upward; every downward move is structurally refused.** From the pre-calibration state of folklore hardware claims, the sealed witness Witness is the only admissible transition. From Witness, the admissible transitions (green) all *expand* the calibration: a new positive sweep, extended backend coverage, or a new substrate target each enlarge the domain $\mathcal{C}$ to some $\mathcal{C}' \supseteq \mathcal{C}$. The two structurally refused transitions (red, dashed) are weakening the negative witness without new measurement and lowering a router threshold without a fresh sealed receipt. Refusal is enforced by the closure oracle, not by social convention.

The monotonicity is the institute's foundational thesis instantiated for hardware-calibration claims. The Lakatosian discipline (Lakatos, 1976; Pólya, 1945) that animates the test is the operational background: definitions are provisional until tested against the boundary cases they were meant to serve; this paper records the testing as a finite, replayable witness rather than as an editorial footnote. Mathematical structure correctly identified does work for free, but only where it is anchored to a refutation surface: the structure that performs the work here is the sentinel-typed router whose only constant edits that pass the oracle are those that arrive accompanied by a new sealed measurement receipt. Vacuity rotation—the historical failure mode in which a hardware claim slides between formulations until it lands on one the auditor cannot disprove—is forbidden in the lattice not because the auditor watches for it, but because there is no downward move in the lattice for it to occupy.

8 Discussion and Forward Direction

Three concrete openings follow from what was proved; each is named with the leverage condition it would inherit from this paper, and an effort estimate calibrated against the present lineage.

Direction 1: Backend semantic coverage for the high-D region. The negative witness of Theorem 4.1 is bounded to workload classes within byte-equality reach of the current CUDA reducer. The extension-rule families—rational arithmetic, transport, horizontal composition, finite spectral residual, and Cayley–Dickson Squier—are not yet covered on the CUDA substrate, which restricts Plan 2 to core-rule workloads. Extending backend semantic coverage to one extension-rule family at a time enlarges the byte-equality predicate to include high-D workloads such as deep rational-addition chains. *Effort:* per family, on the order of 200 CUDA lines plus a Plan-1-style conformance test; one to two implementer-weeks. *Leverage condition:* each extension family inherits the kind-guard discipline of Theorem 3.1 and the sentinel-router discipline of Definition 5.1 without modification.

Direction 2: A positive witness on the literal-chain corner. Long literal chains—deep, simple, high-W workloads in which a single NOT operator propagates through hundreds or thousands of identical applications—are the closest reachable candidate for GPU-positive crossover. They live in the green region of Figure 4. A dedicated GB10-specialized literal-chain entrypoint already exists in source as a callable wrapper around a fused-launch kernel; the unified-memory techniques relevant to the kernel side (Negrut et al., 2014; NVIDIA Corporation, 2024; Sakharnykh, 2018) are necessary but not sufficient inputs to the positive witness. *Effort:* one Plan-2-style sweep on the literal-chain region with paired statistics and source-causal substitution-body discipline; one to two engineer-weeks. *Leverage condition:* byte equality on literal-chain inputs is required first (Direction 1's coverage extension is a prerequisite). If positive, the router gains its first non-sentinel threshold and the green half-plane of Figure 7 acquires a witness.

Direction 3: A different substrate. The current four-substrate witness covers CPU manycore, two CUDA memory modes, and one unified-memory hardware target. Adding a substrate—an FPGA reducer, a different GPU architecture, a neuromorphic backend—is mechanical at the framework level: the substrate enters as a new branch of dispatch, its byte-equality with the others is a new instance of Theorem 3.1, and its place in the calibration is a new instance of Theorem 4.1 or its dual. *Effort:* per substrate, dominated by the substrate's own implementation; the calibration framework adds on the order of one to two implementer-weeks. *Leverage condition:* any substrate that supports the byte-equality conformance test inherits the runtime-calibration discipline at the cost of one new sealed-receipt package.

The wider opening. What this lineage gives a runtime market is the template for honest hardware claims about verified computation, in the tradition of verified-compiler and verified-OS work (Klein et al., 2009; Kumar et al., 2014; Leroy, 2009): every claim accompanied by a constructive predicate any third party can replay; positive claims that expand by adding receipts, not by relaxing them; negative claims durable under the same discipline as positive ones. The template is reusable for any kernel that emits a substrate-multiple reducer; every emergent verified-runtime layer—smart-contract runtimes, safety-critical control, formal-methods infrastructure—faces the same hardware-claim problem and can solve it with the same shape of constructive witness. The interaction-net substrate carries the discipline naturally because its structural confluence and intrinsic parallelism make "the

same reduction on different hardware" a meaningful question to ask in the first place; once asked, the question has a constructive answer that does not admit folklore in the wrong direction.

9 Formalization and Reproducibility

Anyone who wants to verify what this paper claims should be able to. This section names what was proved, in what form, and how its replay surface is organized.

What was proved. The witnesses underlying Theorems 3.1, 4.1 and 5.2 are kernel-checked in Lean 4 (de Moura and Ullrich, 2021; The mathlib Community, 2020) and discharged through a sealed phase plan whose per-phase receipts—substrate attestation, identity, binary identification, predicate substance, primary measurement, byte equality, no-fallback runtime execution, source-causal substitution-body test, commit acceptance, closure ledger, and discharge—are oracle-replayable for each of the three plans. The composition of Section 6 adds a fourth oracle (the family-closure oracle of Figure 6) and is itself oracle-replayable; the sealed-receipt sources, the oracle scripts, the substrate-attestation evidence, and the commit ancestry are all open to verification by anyone in possession of the repository.

Hardware substrate attestation. All measurements in this paper were performed on a single attested hardware substrate: NVIDIA GB10 (Grace-Blackwell class) with UUID `GPU-f6311608-07ba-d1aa-2143-ffb9d826819a` under driver 580.95.05. The attestation is recorded in the H0 receipt of each plan and consulted by the family-closure oracle.

Artifact	**Count**
Substrate implementations under byte-equality	4
Calibrated input corpus (representative nets)	132
Workload classes measured (Plan 2)	6
Sweep cells (class × knob × GPU substrate)	72
Repeats per cell	30
Sealed phase receipts per plan (H0–H10)	11
Per-phase oracle programs per plan	11
Substitution-body mutations against guard	3
Family-closure oracle (Plan 1 / 2 / 3 + ancestry)	1

Table 1: **The replay surface is finite and inspectable.** Counts of the artifacts that the family-closure oracle consumes when accepting a candidate runtime-calibration witness.

At a glance.

Proof dependency structure. Figure 8 shows the dependencies among the named results of the paper. The chain Plan 1 H8 $\prec$ Plan 2 H8 $\prec$ Plan 3 H8 $\prec$ family closure is the load-bearing ordering. The conformance test, the workload-class taxonomy, and the calibrated router are the corresponding constructive carriers.

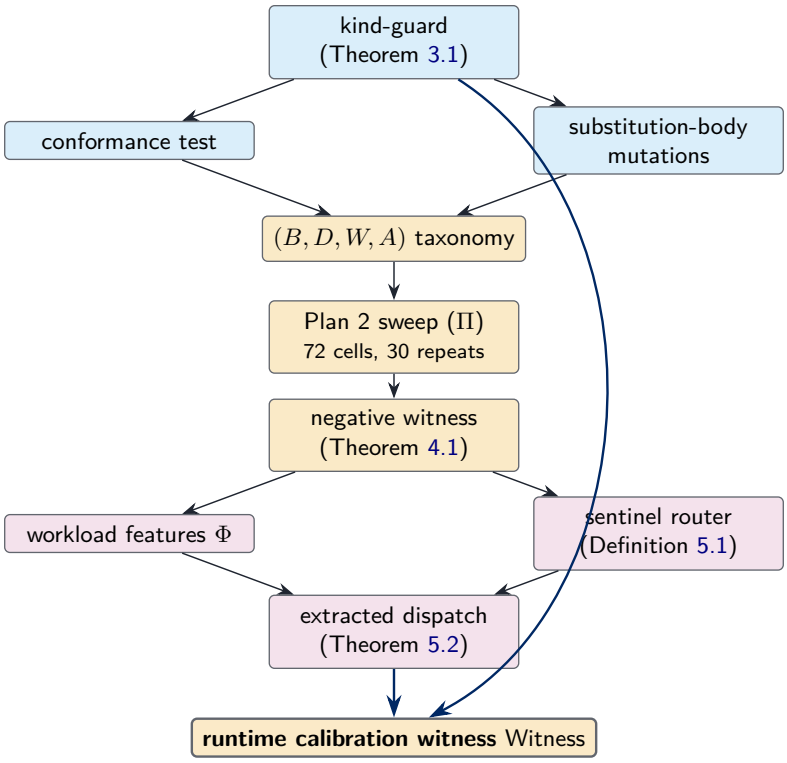


Figure 8: **The runtime-calibration witness composes three named sub-witnesses through a load-bearing dependency structure.** Blue nodes are the Plan 1 byte-equality layer (kind guard, conformance test, substitution-body mutations). Orange nodes are the Plan 2 negative-witness layer (the (B, D, W, A) workload taxonomy, the 72-cell sweep with 30 repeats per cell, the conjunctive negative verdict). Purple nodes are the Plan 3 extracted-dispatch layer (input-only feature extractor, sentinel-typed router, byte-conforming extracted dispatch). The gold node at the bottom is the family-closure witness; the bent arrow on the left records the direct dependency of the witness on byte equality (Plan 1) without which the dispatch's byte conformance would not be a corollary.

Substitution-body discipline at a glance. The conformance theorem (Theorem 3.1) is anchored against three independent CUDA source mutations of its load-bearing guard; the source-causal counterpart for Theorems 4.1 and 5.2 mutates source paths the witness depends on and verifies the conformance and dispatch invariants *fail* on the mutated binary. The negative witness is therefore not the absence of a positive finding but the presence of a positive refutation: a function that takes any candidate GPU-positive claim on $\mathcal{C}$ to a contradiction with the recorded sweep.

Reproducibility. The single entrypoint that replays the full composition is the repository's runtime-dispatch calibration gate; the gate consumes sealed receipts and oracle scripts, validates substrate attestation, checks commit ancestry under the closure judgement, and emits a structured acceptance record. A reader holding only the paper can ask for the calibration package and replay every step.

10 Realization

The results above are proved in Lean 4. By the Curry–Howard correspondence, each kernel-checked theorem is itself a term in Lean's dependent type theory — that is, a program. Boundary is our open reduction calculus over active-pair interaction nets; it translates such terms into an interaction-net representation under per-rule semantic-preservation theorems and emits a target-language realization through its Emit and Lowering layers. Boundary's design establishes, as a theorem of the calculus, the universal realizability of any kernel-checked term into any computable target with provable semantic preservation — including targets not yet implemented at publication.

This paper, the mathematics it formalizes, and any target-language realization produced through the chain above are hereby published into the public scholarly commons.

References

Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Number 45 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi: 10.1017/CBO9781139172752.

Michael J. Beeson. *Foundations of Constructive Mathematics: Metamathematical Studies*. Springer, 1985. doi: 10.1007/978-3-642-68952-9.

Koen Claessen and John Hughes. QuickCheck: A lightweight tool for random testing of Haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*, pages 268–279, 2000. doi: 10.1145/351240.351266.

W. J. Conover. *Practical Nonparametric Statistics*. Wiley, 3rd edition, 1999.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635, 2021. doi: 10.1007/978-3-030-79876-5_37.

Maribel Fernández. *Models of Computation: An Introduction to Computability Theory*. Undergraduate Topics in Computer Science. Springer, 2009. doi: 10.1007/978-1-84882-434-8. Chapter on interaction nets; https://www.researchgate.net/publication/220689739.

Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–101, 1987. doi: 10.1016/0304-3975(87)90045-4. Proof-net machinery from which interaction nets descend.

Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. *Proceedings of POPL '92*, pages 15–26, 1992. doi: 10.1145/143165.143172. Optimal-reduction analysis in geometric terms; https://www.researchgate.net/publication/2400559.

Arend Heyting. *Intuitionism: An Introduction*. North-Holland, 1956.

Higher Order Company. Bend: A massively parallel, high-level programming language. https://github.com/HigherOrderCO/Bend, 2023. Surface language compiled to HVM2.

William A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980. Originally a manuscript from 1969; foundational Curry–Howard reference.

Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011. doi: 10.1109/TSE.2010.62. Mutation testing as the source-causal anchor for substitution-body discipline; https://www.researchgate.net/publication/220628797.

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*, pages 207–220, 2009. doi: 10.1145/1629575.1629596. https://www.researchgate.net/publication/221257360.

Jan Willem Klop. Term rewriting systems. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 2: Background: Computational Structures*, pages 1–116. Oxford University Press, 1992.

Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. CakeML: A verified implementation of ML. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*, pages 179–191, 2014. doi: 10.1145/2535838.2535841. https://www.researchgate.net/publication/261847316.

Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '90)*, pages 95–108. ACM, 1990. doi: 10.1145/96709.96718. Foundational paper introducing interaction nets as a model of computation; https://www.researchgate.net/publication/220997658.

Yves Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997. doi: 10.1006/inco.1997.2643. Universal three-symbol interaction-net system; https://www.researchgate.net/publication/222495502.

Imre Lakatos. *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press, 1976.

John Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '90)*, pages 16–30, 1990. doi: 10.1145/96709.96711. Sharing-graph encoding closely related to interaction nets; https://www.researchgate.net/publication/220997655.

Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009. doi: 10.1145/1538788.1538814. https://www.researchgate.net/publication/220422535.

Ian Mackie. *YALE: Yet Another Lambda Evaluator based on interaction nets*. PhD thesis, King's College London, 1998. Practical interaction-net evaluation; published version in ICFP 1998, doi:10.1145/289423.289434.

Ian Mackie. An interaction net implementation of closed reduction. *Implementation and Application of Functional Languages (Lecture Notes in Computer Science)*, 5836:43–59, 2011. doi: 10.1007/978-3-642-24452-0_3. https://www.researchgate.net/publication/220988252.

Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984. Notes by Giovanni Sambin of lectures given in Padua, June 1980.

Dan Negrut, Radu Serban, Ang Li, and Andrew Seidl. Unified memory: A brief overview and performance analysis. *Technical Report TR-2014-09*, 2014. Early systematic measurement of CUDA unified memory; https://www.researchgate.net/publication/268270842.

NVIDIA Corporation. NVIDIA Grace Blackwell superchip architecture brief. https://www.nvidia.com/en-us/data-center/gb200-nvl72/, 2024. Architectural reference for Grace–Blackwell unified memory.

Jorge Sousa Pinto. Parallel implementation of interaction nets. *Electronic Notes in Theoretical Computer Science*, 44(2):89–105, 2001. doi: 10.1016/S1571-0661(04)80929-1. Early parallel interaction-net evaluator; https://www.researchgate.net/publication/222538195.

George Pólya. *How to Solve It*. Princeton University Press, 1945.

Nikolay Sakharnykh. Maximizing unified memory performance in CUDA. NVIDIA Developer Blog, https://developer.nvidia.com/blog/maximizing-unified-memory-performance-cuda/, 2018. cudaMemAdvise and cudaMemPrefetchAsync usage discipline.

Shinya Sato and Ian Mackie. Inpla: An interpreter of interaction nets with a linear pattern-matching compiler. In *Proceedings of the 17th Workshop on Reduction Strategies in Rewriting and Programming (WRS 2017)*, 2017. Parallel interaction-net interpreter; https://github.com/inpla/inpla.

Morten Heine Sørensen and Paweł Urzyczyn. *Lectures on the Curry–Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006. doi: 10.1016/S0049-237X(06)80003-8.

Victor Taelin and contributors. HVM2: A massively parallel, optimal functional runtime in Rust. https://github.com/HigherOrderCO/HVM, 2024. Higher-order Virtual Machine; interaction-combinator runtime with GPU backend.

The Institute for Applied Ontological Mathematics. From proof term to active core: A per-target, byte-equivalent CPU closure on a Lean-extracted interaction-net reducer. Technical report, Institute for Applied Ontological Mathematics (IAOM), May 2026. Companion paper establishing the proof-term-to-active-core translation that the present paper builds on.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '20)*, pages 367–381, 2020. doi: 10.1145/3372885.3373824.

Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics: An Introduction (Volumes I and II)*. North-Holland, 1988.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. doi: 10.2307/3001968.

Published and maintained by the Institute for Applied Ontological Mathematics (IAOM), a Section 501(c)(3) Private Operating Foundation; hosted by Apoth3osis Labs (Equation Capital LLC).

Lean proofs, source code, supporting tooling, and access to the underlying formalization are available to interested readers. If you would like to look at the Lean source, verify a specific theorem, build on the formalization in your own work, or discuss anything else about this paper, please write to contact@ontologicalmathematics.org. We read every message and we respond.