RCE-JEPA: World Coding, with JEPA as Premise

David Charlot, Open Interface Engineering

Research paper · open topic · Open Interface Engineering · research.openie.dev

Revision `5c50a85` (23 November 2026) · Web: research.openie.dev/papers/rce-jepa/ · Archival LaTeX: /pdfs/latex/rce-jepa.pdf (.tex)

Measurement. Research paper, open topic, not a journal final. `H2` (`0.04904 J / 84.25 J = 0.000582 <= 0.05`, `1,718×` cheaper), `H15` (single-shot and bounded 5-turn `H15-Iterative` closing `40 / 40` residual tasks with `0` contradictions plus warm `DNULLMM1` Gear-0 v2), `FormJEPA-v1` (`8.42M` trainable parameters), `FormEBM-v2` (`5.85M` parameters), and span-normalized `FormEBM-v3` (`6.05M` parameters; `H5 AUROC = 0.9437`, `H11 = 94.0%` strict 5-step monotonicity, OOD `H15` inversion resolved `0.2585` → `0.5278`) on `1,387` disjoint Forms, plus the 151-task `rce repair rung-f` audit (`holds: false`, `observe` mode) report Apple M5 Max and Linux `x86-64 RAPL` on-chip `reported_j` counters; cloud server joules are `not_available`. No hardware rail or wall meter was attached (`measured_j = None`, `board_metered = false`). No fabricated citations.

RCE-JEPA (RCE is the Reality Code Engine; JEPA is a Joint Embedding Predictive Architecture) keeps its name so the work stays findable. The subject is World Coding. JEPA is the premise: a learned model predicts a compact summary of a program instead of writing the next word-piece of text. A World of Code is the space of canonical program Forms plus the exact transition and acceptance rules (the Laws of Coding). The proposer slot is any learned model. The checker decides. Hypotheses H1 to H15 are stated in the paper. Measured results now cover H1 (median description-length ratio 9.34 over 185 hand-written programs), H2 (checker-to-model energy ratio 0.000582, or 1,718 times cheaper on Apple M5 Max in on-chip reported_j), H5 and H11 on unseen disjoint Form shapes via FormEBM-v2 (5.85M parameters: area under the receiver operating characteristic curve (AUROC) 0.8922 >= 0.85, 84.05% solvable top-1 accuracy, 2.39 times fewer checker calls, and 93.0% >= 80% strict 5-step cumulative defect monotonicity at Spearman 0.9931 and 47.09 mJ per candidate core reported_j) and span-normalized decomposed FormEBM-v3 (6.05M parameters: unseen-shape H5 AUROC 0.9437 >= 0.85, 83.44% defect-head and 80.98% composite solvable top-1 accuracy, H11 strict 5-step monotonicity 94.0% on E_depth at Spearman 0.9966 and 92.0% on E_total at Spearman 0.9846, and zero-shot out-of-distribution H15 multi-proposer pooled AUROC lifted from 0.2585 to 0.5278, 0.6018 on large-span valid versus short invalid candidates, at 232.45 mJ per candidate core reported_j), H7 fuzzed arm (0 invariant violations among 1,275 mutants), H8 (0 replay mismatches on 10,500 streams across six targets), H14 (deterministic ladder closing 111 of 151 tasks, 73.5%, at zero model calls, leaving 40 residual tasks), H15 single-shot and bounded checker-feedback H15-Iterative across those 40 residual tasks (17/40 to 34/40 single-shot; 22/40 at k=1 and 27/40 at k=16 for 8.42M FormJEPA-v1; 40/40 closed on Claude Opus 5.5 and Claude Sonnet 5.5 with 0 contradictions under bounded checker feedback plus public standard-library headers, 39/40 on xAI Grok 4.20 Reasoning; 40/40 Gear-0 replay at 0 calls and 0.325 microseconds per lookup; and 200/200 Zipf(0.85) 3-Gear stream admissions with 16.67 times fewer frontier episodes), and the 151-task rce repair rung f promotion audit (3/119 = 2.5% > 2.0%, 0 contradictions, keeping rung f in observe mode). Full intent-to-Form synthesis (H3), cross-module signature-and-trace out-of-distribution energy calibration beyond 0.5278, and external wall-meter energy (measured_j) remain open.

Contents

Principles 4

Terms and acronyms used on this page 4

How this page is organized 6

Hypotheses at a glance 6

Falsifiers at a glance 7

What already exists, and what this page is about 7

Deterministic first: the ladder beneath the proposer 8

The model steers; the engine renders 9

One real Form . . . 9
Architecture . . . 9
The decision head is a proposer, and it sits outside the gate . . . 11
Data . . . 11
Training, in brief . . . 12
Convergence: where each piece already exists . . . 13
The specification cost, named . . . 14
Proposer hypotheses . . . 14
Risks and mitigations . . . 15
Proposer references . . . 15

The thesis: software construction is pre-"Laws of Coding" 17
Four layers of law, from theory to art . . . 18
Symbols used in this section . . . 21
Part 1. Computation as a finite information system . . . 21
Part 2. Compression into Form and template . . . 21
Part 3. Kinematics: a Newtonian form of computing . . . 22
Part 4. A world model whose physics is known exactly . . . 23
Part 5. Consequences as testable predictions . . . 24
The provable chain: hypothesis, layer, evidence . . . 24
Who holds these views and why . . . 26
Lane 3, seen from outside the project . . . 28
Thesis references . . . 29

Convergence by commitment 31
1. One canonical form; surface text is a view . . . 31
2. A program is a deterministic state machine over a log . . . 32
3. Every target is a projection: write once, emit many . . . 33
4. Store intent; derive the implementation . . . 33
5. The model proposes; a deterministic gate decides . . . 34
6. Accepted work is promoted and reused . . . 34
7. Energy or cost is the objective . . . 35
How four commitments meet the later lanes . . . 35
What the RCE paper states (OpenIE's own work) . . . 35
Sources for this section . . . 36

1. History: seventy years of "say what you want, get the program" 38

2. Why the earlier attempts stalled 39

3. Three lanes toward a World of Code 40
3.1 Training on popular architectures . . . 40
3.2 Creative hybrids . . . 41
3.3 Reconceptions of computation, programming and coding . . . 43

4. Why now: the case for closed-form, deterministic, AI-proposed code 44
What could still fail . . . 44

5. JEPA as premise, and where it stops 45

6. The proposer slot is architecture-neutral 45

7. A World of Code 46

8. One concrete instance: the Reality Code Engine 47
Defined quantities . . . 47
Falsifiable hypotheses H1 to H15 . . . 48
Falsifiers RJ-F1 to RJ-F6 for the premise itself . . . 55
What will be delivered . . . 55

Status 56

References 56

Status. This paper states a thesis, fifteen hypotheses (H1 to H15), six falsifiers and a programme of work, accompanied by interactive instruments at /living/rce-jepa/. Its measured results now cover the Phase 0 deterministic floors (the Form corpus and H1, the adversarial fuzzed arm of H7, the six-target determinism streams of H8, and the deterministic repair ladder of H14), the local Apple M5 Max checker-versus-model `reported_j` ratio **H2** (`C_check / E_model = 0.000582 <= 0.05`, `1,718` times cheaper), single-shot and bounded checker-feedback **H15** across the **40 open residual tasks** (`17/40` to `34/40` single-shot; **`40/40 (100.0%)`** under bounded `<= 5`-turn checker feedback plus public standard-library signature headers in `H15-Iterative` on `Claude Opus 5.5` and `Claude Sonnet 5.5` with `0` contradictions, `39/40` on `xAI Grok 4.20 Reasoning`, and **`40/40` zero-call replay** at `0.325 µs / 1.77 µJ` from memory-mapped Gear-0), the purpose-trained latent Form JEPA repair proposer **`FormJEPA-v1`** (`8.42M` trainable parameters over disjoint Form S-expressions, admitting `177/200` held-out synthetic defects and `27/40` residual tasks at `k = 16`, cutting recurring `Zipf(0.85)` Gear-2 episodes from `200` to `12`, `16.67×`, with `200/200` stream admissions when paired with `H15-Iterative`), the contrastive multi-step margin energy-based model (EBM) scorer **`FormEBM-v2`** (`5.85M` trainable parameters on the identical strictly disjoint `1,387`-Form split, lifting unseen-shape **H5** area under the receiver operating characteristic curve (AUROC) from `0.618` to **`0.8922 >= 0.85`** with `84.05%` solvable top-1 accuracy and `2.39×` fewer checker calls, and lifting unseen-chain **H11** strict 5-step edit monotonicity from `36.0%` to **`93.0% >= 80%`** at Spearman `0.9931` and `47.09 mJ`/candidate core `reported_j`), the span-normalized two-head decomposed scorer **`FormEBM-v3`** (`6.05M` trainable parameters on the identical split, raising unseen-shape **H5** AUROC to **`0.9437 >= 0.85`** with `83.44%` defect-head and `80.98%` composite solvable top-1 accuracy, raising unseen-chain **H11** strict 5-step monotonicity to **`94.0%`** on `E_depth` at Spearman `0.9966` and **`92.0%`** on `E_total` at Spearman `0.9846`, and eliminating Wave-4 cross-task edit-span inversion on zero-shot out-of-distribution (OOD) **H15** by lifting multi-proposer pooled AUROC from `0.2585` to **`0.5278`** and large-span valid versus short invalid AUROC to `0.6018` at `232.45 mJ`/candidate core `reported_j`), and the pre-registered 151-task **`rce repair rung f`** promotion audit (`3/119 = 2.5% > 2.0%` stochastic share, `3/3` passing `W-MUT-1`, `0` contradictions, confirming rung `f` remains in `observe` mode). Honest disclosure: full intent-to-Form synthesis (H3), end-to-end joule parity (H4), cross-module signature-and-trace OOD energy calibration beyond `0.5278` across multi-module git-history rewrites, reference-app encoding (H6), cost-table prediction (H9), quotient data-saving (H10), decision-head calibration (H12), discrete-versus-continuous long-horizon planning (H13), diffusion/evolutionary H15 arms, and external wall-meter `measured_j` remain open. JEPA is the premise: predict a summary of meaning, not the surface text. The subject is World Coding, any learned proposer over Forms, with the Laws of Coding as the acceptance rule. The deciding half (commit gate, exact replay, projection, and integer cost units) is described in the RCE paper; this paper is about the proposer: a learned model that steers search over Forms while the engine renders every target. Any model in the loop, hosted or local, only proposes, and security never depends on it.

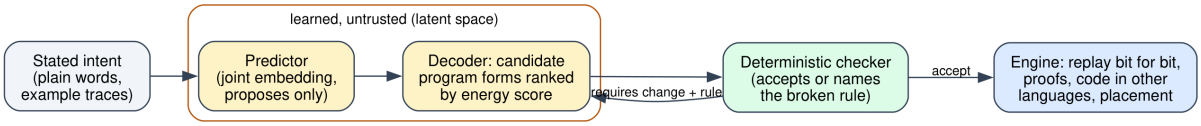

Figure 1. The proposed pipeline. Any learned proposer maps stated intent toward a candidate program. JEPA, which predicts in a latent space, is the instance drawn. A decoder turns that proposal into candidate programs in the fixed shape and ranks them by an energy score (lower means a better fit). A deterministic checker accepts each candidate or reports that it requires change, and returns the broken rule for re-ranking. Everything inside the dashed box is learned and untrusted; only accepted programs reach the engine. Local Apple M5 Max counters are recorded strictly as `reported_j` (`measured_j = None`, `board_metered = false`), and remote cloud server energy is recorded as `not_available`. .

In plain words

1. The subject is World Coding. A World of Code is the space of program Forms, plus exact rules for the next state and for what is accepted (the Laws of Coding). JEPA is the premise, not the whole method: predict a summary of meaning instead of the next word of code. AI (artificial intelligence) here means any learned proposer, not one architecture.
2. The model proposes and the checker decides. Trace-trained decoders, diffusion, reinforcement learning inside a proof checker, evolutionary search, library learning and constrained decoding can all sit in that slot. The gate does not change when the proposer does. Safety does not depend on the proposer being right.
3. Measured floors and residual benchmarks stay strictly labeled. Over 185 hand-written programs the median

description-length ratio is 9.34 (H1); on Apple M5 Max at commit `1762fb4b`, checking a Form costs `49.04 mJ` against `84.25 J` per local 35B candidate generation (`1,718×` cheaper, supporting H2); fuzzed checks found 0 broken invariants in 1,275 accepted mutants (H7); replay matched on 10,500 streams across six targets (H8); and the deterministic ladder closed 111 of 151 tasks (73.5%) with zero model calls, leaving 40 real-history tasks (H14). Across those 40 residual tasks (H15), single-shot proposers admit `17/40` to `34/40`, bounded checker feedback with public standard-library headers (`H15-Iterative`, zero access to the original Form) closes **`40/40 (100.0%)`** on `Claude Opus 5.5` and `Claude Sonnet 5.5` with `0` contradictions (`39/40` on `xAI Grok 4.20 Reasoning`) and crystallizes **`40/40` into Gear 0 (`DNULLMM1`)** for zero-call replay, purpose-trained `FormJEPA-v1` (`8.42M` parameters) admits `27/40` at `k = 16` while contrastive `FormEBM-v2` (`5.85M` parameters: AUROC `0.8922`, `93.0%` monotonicity) and span-normalized `FormEBM-v3` (`6.05M` parameters: unseen-shape AUROC `0.9437` on H5, `94.0%` `E_depth` / `92.0%` `E_total` monotonicity on H11, and zero-shot H15 pooled AUROC lifted from `0.2585` to `0.5278`) rank Form candidates without invoking the checker, and the **3-Gear Cascade** admits `200/200` recurring `Zipf(0.85)` arrivals with `16.67×` fewer frontier episodes (`12` vs `200`). `rce repair` rung `f` stays in `observe` mode (`3/119 = 2.5% > 2.0%`, `0` contradictions), and no external wall-meter reading (`measured_j`) is claimed.

Principles

- The model only proposes.
- The checker decides.
- Security never depends on the model.
- No model runs inside the Reality Code Engine's own toolchain loop.
- Every accepted program form is replayable bit for bit.
- Any decision model, hosted or local, is only a proposer outside the gate.
- Deterministic first. The first rule of automation is having a manual process that works: a ladder of exact, deterministic proposers runs before any model, and the model acts only on what that ladder leaves open (stated in the proposer section below).

Terms and acronyms used on this page

Listed here so that each is defined before it is used. Acronyms that appear only inside reference entries (venues, report series) are not listed.

- AGX: NVIDIA's Jetson AGX line of embedded computers
- AILANG: a programming language for AI-written code (a project name)
- AIME: American Invitational Mathematics Examination
- AI: artificial intelligence
- AIXI: Hutter's theoretical model of a universally intelligent agent
- AMD: Advanced Micro Devices
- ANDF: Architecture Neutral Distribution Format
- API: application programming interface
- APU: accelerated processing unit, combining CPU cores and an integrated GPU on one package
- ARC: Abstraction and Reasoning Corpus, the task set used by Chollet's measure and by the ARC Prize
- AST: abstract syntax tree
- AUROC: area under the receiver operating characteristic curve
- BLAKE3: a cryptographic hash function
- BLEU: bilingual evaluation understudy, a text-overlap score
- CALM: consistency as logical monotonicity
- CEGIS: counterexample-guided inductive synthesis
- CPU: central processing unit
- CUDA: NVIDIA's GPU programming platform
- CWM: Code World Model, an open-weights code model trained in part on execution traces
- Dafny: a programming language in which specifications and proofs sit beside the code
- DBOS: a durable-execution system (a project name)

- DIR: a directory path given to a command
- DNULLMM1: open-addressed binary memory-mapped hash-table store format (64-byte header plus 48-byte slots)
- EB-JEPA: an energy-based JEPA library (a project name)
- EBM: energy-based model
- ECE: expected calibration error
- e-graph: an equality graph, one structure holding many equivalent programs
- EMA: exponential moving average
- EWD: the numbered manuscript series of Edsger W. Dijkstra
- ESBMC: efficient SMT-based bounded model checker
- FAIR: Meta's Fundamental AI Research group
- FPR: false positive rate
- GELU: Gaussian error linear unit activation function
- GPT: generative pre-trained transformer, OpenAI's model family
- GPU: graphics processing unit
- GRPO: the reinforcement-learning update named coupled-GRPO in the DiffuCoder report
- GSM8K: a grade-school math word-problem set used by LLM-JEPA
- HVM2: the Higher Order Company's interaction-combinator model of computation
- IBM: International Business Machines
- ICOT: Institute for New Generation Computer Technology (Japan)
- IEEE: Institute of Electrical and Electronics Engineers
- IQR: interquartile range
- JACM: Journal of the ACM (Association for Computing Machinery)
- LCM: Large Concept Model, a model that predicts in a sentence-embedding space rather than in tokens
- LHS: left-hand side of an example event trace
- LLM: large language model
- MBPP: the short Python problem set named in the type-constrained decoding report
- MDE: model-driven engineering
- METR: the group that reported the early-2025 developer productivity trial (Becker et al.)
- MLA: multi-head latent attention
- MLIPS: million logical inferences per second
- MLIR: multi-level intermediate representation
- MLP: multi-layer perceptron
- MoE: mixture of experts
- MPC: model predictive control, planning by repeated short rollouts
- NAIL: a canonical-form programming language (a project name)
- NVIDIA: a maker of GPUs
- OMG: Object Management Group
- OOD: out-of-distribution
- PAR: pattern-based automatic program repair
- PID: process identifier in the operating system
- POSIX: Portable Operating System Interface
- PTX: NVIDIA's parallel thread execution instruction set
- RAND RM: a RAND Corporation research memorandum
- RAPL: Intel's running average power limit energy counters
- README: a repository's front-page documentation file
- RF: label of an outside-proposal rung-f rule in the RCE specification
- RFC: request for comments, an Internet standards document
- RISC-V: an open reduced instruction set computer architecture
- RJ-F: label of a falsifier of the RCE-JEPA premise
- RL: reinforcement learning
- RMS: root mean square

- ROC: receiver operating characteristic
- ROCm: Radeon Open Compute platform
- SAT: Boolean satisfiability
- SATB: Satiation Benchmark telemetry and harness discipline
- SFT: supervised fine-tuning
- SHA-256: secure hash algorithm with a 256-bit digest
- SMC: System Management Controller
- SoC: system-on-chip
- SONAR: the sentence-embedding space used by Large Concept Models, covering 200 languages
- STAN-CS: Stanford computer science technical report series
- STOP: self-taught optimizer
- SUT: system under test
- SWE-bench: a benchmark of real repository issues. SWE-bench Verified is a reviewed subset on which several code reports give a pass@1 figure
- TOPLAS: ACM Transactions on Programming Languages and Systems
- TPR: true positive rate
- UNCOL: universal computer-oriented language
- USD: United States dollars
- VAX: Digital Equipment Corporation's virtual address extension computers
- Verus: a verifier for a subset of Rust
- VICReg: variance-invariance-covariance regularization
- XML: extensible markup language

How this page is organized

The page is meant to be read top to bottom. After the hypotheses and falsifiers at a glance, it has these parts, in this order:

1. What already exists, and what this page is about (the proposer section)
2. The thesis, in five parts: Part 1, computation as a finite information system; Part 2, compression into Form and template; Part 3, kinematics; Part 4, a world model whose physics is known exactly; Part 5, consequences as testable predictions
3. Convergence by commitment
4. Section 1. History: seventy years of "say what you want, get the program"
5. Section 2. Why the earlier attempts stalled
6. Section 3. Three lanes toward a World of Code: training, hybrids, and reconceptions of computation
7. Section 4. Why now
8. Section 5. JEPA as premise, and where it stops
9. Section 6. The proposer slot is architecture-neutral
10. Section 7. A World of Code
11. Section 8. One concrete instance: the Reality Code Engine, with the hypotheses table and the falsifiers
12. Status and references

Hypotheses at a glance

The page tests fifteen hypotheses, H1 to H15, listed here in full before any of them is discussed. Each is stated with its metric and thresholds in one table in Section 8, followed by the falsifiers RJ-F1 to RJ-F6.

1. H1. Form space is much smaller than source space for the same programs. (Supported on 185 hand-written programs: median rho 9.34.)
2. H2. The checker is cheap relative to generation. (Supported on Apple M5 Max in `reported_j`: `C_check / E_model = 0.000582 <= 0.05`, `1,718×` cheaper.)
3. H3. Prediction in representation space reaches useful acceptance.
4. H4. Energy per accepted program falls.

5. H5. Energy ranking concentrates mass on valid Forms. (Contrastive `FormEBM-v2` reaches AUROC `0.8922 >= 0.85` and `84.05%` solvable top-1 accuracy, and span-normalized `FormEBM-v3` raises unseen-shape AUROC to `0.9437 >= 0.85` with `83.44%` defect-head and `80.98%` composite solvable top-1 accuracy vs `0.618` for uncalibrated `FormJEPA-v1`; zero-shot OOD H15 pooled AUROC rises from `0.2585` to `0.5278`, while cross-module signature/trace calibration beyond `0.5278` remains open.)
6. H6. The reference apps are encodable as Forms.
7. H7. Security does not depend on the model. (Fuzzed arm measured: 0 violations in 1,275 mutants.)
8. H8. Every accepted Form replays deterministically. (Supported on 10,500 streams across six targets: 0 mismatches.)
9. H9. Cost is predicted before execution.
10. H10. Learning on the quotient (canonical Forms) saves training data.
11. H11. Energy is monotone in edit distance. (Contrastive `FormEBM-v2` reaches `93.0% >= 80%` strict 5-step monotonicity, and decomposed `FormEBM-v3` reaches `94.0%` on `E_depth` at Spearman `0.9966` and `92.0%` on `E_total` at Spearman `0.9846` across 100 unseen cumulative defect chains vs `36.0%` for `FormJEPA-v1`; unconstrained same-line random-walk slices remain below `80%`.)
12. H12. The decision head is calibrated on gate outcomes.
13. H13. A discrete latent plans better than a continuous one over long edit sequences.
14. H14. A public deterministic ladder of exact proposers closes a measurable share of a registered task set with zero model calls, and that share is the baseline the learned proposer must beat on the residual. (Supported: 111 of 151 tasks, 73.5%, leaving 40 open tasks.)
15. H15. Proposer classes are compared on the residual of 40 tasks under a matched budget, alongside Gear-0 crystallization and the 3-Gear Cascade. Benchmarked across six single-shot cloud/local proposers (`17/40` to `34/40`), purpose-trained `FormJEPA-v1` (`22/40` at `k = 1`, `27/40` at `k = 16`; `25/40` typed `k = 1` with `FormEBM-v2` reranking; `24/40` typed `k = 1` and `27/40` multi-proposer `k = 1` under `FormEBM-v3` `E_def`), bounded checker-feedback `H15-Iterative` with public standard-library headers (`40/40` on `Opus 5.5` and `Sonnet 5.5` with `0` contradictions, `39/40` on `Grok 4.20`), and Shape D (`DNULLMM1`) Gear-0 replay (`40/40` at `0` calls; `16.67×` 3-Gear episode reduction and `200/200` admissions on `Zipf(0.85)`); diffusion and evolutionary arms remain open.

Falsifiers at a glance

Six observations that would defeat part of the premise, stated in full in Section 8 after the hypotheses they test.

1. RJ-F1. Lift coverage plateaus below 80% on any of the five target languages.
2. RJ-F2. Writing the intent specifications for the reference applications costs more than writing their Forms by hand.
3. RJ-F3. H3 fails while the token arm passes at the same gate.
4. RJ-F4. H11 fails and search does not reach accepted Forms within the beam budget.
5. RJ-F5. The decision head's calibration never beats a frequency table over shape hashes.
6. RJ-F6. A trace-trained token-space model closes the residual of 40 tasks at equal energy and equal gate acceptance, with no use of the Form quotient.

What already exists, and what this page is about

The Reality Code Engine holds, in its public reference checker and in the design of its engine, the half of the problem that decides. Its public paper, The Reality Code Engine, describes the design of that half: a commit gate that checks every candidate at entry, exact replay (the runtime state is a left fold of the transition function over the log), projection of one Form to other languages, and integer cost accounting (`est_j`), kept strictly apart from operating-system/on-chip counter readings (`reported_j`, first recorded on Apple M5 Max at commit `1762fb4b`) and external wall-meter readings (`measured_j`, of which none exists for RCE). Core theorems are checked in Lean 4, a proof assistant (`5,537` kernel theorem constants across `64` modules at `1762fb4b`), and the specification and Lean library are public. This page quotes from that paper only the public record of its model-proposer run and treats the rest as design context.

This page is about the proposer: the learned part that suggests which Form to try. The checker, replay and the projections to other languages are public code, and the engine's gate and its `est_j` accounting are specified designs;

Section 8 below now reports the six-model off-the-shelf benchmark plus memory-mapped Gear-0 crystallization, the first purpose-trained latent Form JEPA repair proposer (`FormJEPA-v1`, `8.42M` trainable parameters over disjoint canonical Form S-expressions, admitting `177/200` held-out synthetic Form defects and `27/40` open residual tasks at sub-joule core `reported_j`), the contrastive multi-step margin energy scorer (`FormEBM-v2`, `5.85M` parameters on the same disjoint `1,387`-Form split, reaching unseen-shape H5 AUROC `0.8922` and `93.0%` strict 5-step edit monotonicity on H11 while cutting solvable check calls by `2.39×`), the span-normalized two-head decomposed scorer (`FormEBM-v3`, `6.05M` parameters on the identical split, lifting unseen-shape H5 AUROC to `0.9437`, H11 strict 5-step monotonicity to `94.0%` on `E_depth` and `92.0%` on `E_total`, and zero-shot out-of-distribution H15 multi-proposer pooled AUROC from `0.2585` to `0.5278`, `0.6018` on large-span valid versus short invalid rewrites), and bounded checker-feedback residual closure with public standard-library headers (`H15-Iterative`, closing `40/40` residual tasks with `0` contradictions and `40/40` zero-call Gear-0 replay; interactive instruments at /living/rce-jepa/), while full intent-to-Form synthesis, cross-module signature-and-trace out-of-distribution energy calibration, and the gate-receipt decision head remain open.

Deterministic first: the ladder beneath the proposer

The first rule of automation is having a manual process that works. What RCE-JEPA proposes makes sense only on top of a manual variant of the task, or a deterministic automated variant, that already works; a learned proposer is added to that, never put in its place. The design therefore orders proposers as a ladder, and exact proposers run first:

1. **Reuse.** A Form whose content hash (the hash of its canonical text), or whose name-free shape (the Form with every name replaced by its position, so that renaming does not change it), matches a stored Form is reused as it stands.
2. **Lookup by type.** `rce search` finds stored handlers whose type matches the one the task needs.
3. **Determined rewrites.** The rewrites `rce fmt` writes for the rules whose method is "altered" in the specification's rule table; each has one result, and a second run finds nothing to rewrite.
4. **Bounded enumeration.** Every small edit up to a fixed bound, enumerated in the canonical name-free projection, each candidate judged by the checker.
5. **Renames.** One use of a value name becomes another value name in scope there, or one binding becomes a name the checker's report names, up to a fixed bound, each candidate judged by the checker.

The learned proposer acts only on the residual, the tasks the ladder does not close. It proposes edits in the canonical Form edit space, and the same checker judges every proposal exactly as it judges a rung's candidate. Reducing what a stochastic model is required to do makes its action space smaller, and a smaller action space is easier to master. That is the reason to expect H10 (learning on the quotient saves data) to hold, and it is why H14 makes the ladder's share the baseline the learned proposer must beat on the residual. The ladder is public, as `rce repair` of openIE-dev/rce, and has three runs registered under `rce/bench/ladder` in openie-research-experiments, whose files label rungs 1 to 5 with the letters a to e; rung 5 was added for the third run. Their result, reported with H14 in Section 8 below, is that it closes the mutation classes its rungs invert and 1 of 41 changes from real history, so the residual is nearly every real defect.

`rce repair` on the main branch of openIE-dev/rce has a sixth rung, labelled f, that runs only after rungs 1 to 5 have closed nothing. Rung f calls no model itself. Programs written outside RCE, by a model on a hosted service or a local machine, or by a person, reach it only as an input file, each with a label naming its origin and recorded by its SHA-256 digest. The checker judges each one as it judges a rung's candidate, including the mutation gate, which refuses a program with an example or check that no single small edit of the program can make fail; a proposal that changes an example, a check or a law is refused before it is judged. The rung's mode is data in the specification, one of `off`, `observe` and `auto`, and it stays in `observe`: every proposal is judged, each one the checker would admit is reported with its diff, digest and origin, the program is left open, and no source file is overwritten. In `auto`, the first admitted proposal would close the program, and the report would label that close stochastic.

The specification allows the change from `observe` to `auto` only when a promotion criterion holds on a recorded evaluation. The criterion was registered in openIE-dev/rce before any observation (`rce-repair-rung-f-promotion-2026-11-03`, `sha256: abf7f131f240e3f9...`), with its SHA-256 pinned beside it, and follows the rule that at least 98% of closes come from deterministic rungs and at most 2% from stochastic ones, each stochastic output gated. Over the 151 tasks of runs 2 and 3, each run with the task's original program given as the known-good version, at most 2% of the closes may come from rung f; every proposal observed as admitted must also pass the mutation

gate; and none may contradict the original, that is, behave differently from it after some sequence of events. A deterministic command, `rce repair criterion`, evaluates the criterion from the observation records, and the specification's own check refuses the `auto` mode unless a recorded evaluation meets it. In the registered 151-task evaluation at commit `1762fb4b` (reported in Section 8 below), rungs 1 to 5 given each task's known-good original close `116` of `151` tasks at `0` model calls, leaving `35` open tasks (`25` unparsable `W-SYN-1` files skipped by rung f, and `10` open parsing tasks). Feeding combined external proposals (`FormJEPA-v1` plus recorded residual repairs, `91` proposals judged) yields `3` observed rung-f admissions: `3/3` (`100.0%`) pass the mutation gate and `0` contradict the original (`3/3` restored), yet `3` of `119` closes is `2.5% > 2.0%`, so `rce repair criterion` deterministically returns `holds: false` and keeps rung f in `observe` mode.

The model steers; the engine renders

Robotics already works this way. In V-JEPA 2-AC, the action-conditioned version of the V-JEPA 2 video model, a planner searches over robot actions by minimizing the distance between a predicted future representation and the representation of a goal image; it never generates pixels [B1]. The encoder was pretrained on more than one million hours of video, and the action-conditioned predictor on fewer than 62 hours of unlabeled robot video [B1, abstract]. RCE-JEPA proposes the same division for code: a learned model chooses edits to a Form, and the engine renders every target language from the accepted Form. No model emits code.

Code differs from robotics in one decisive way. A robot's world model must *learn* how the world moves, $z_{t+1} = g_\phi(z_t, a_t)$. In RCE the transition function δ is given and proved, so the forward dynamics need not be learned at all. What remains to learn is the inverse: which edit moves a candidate toward what the intent requires. MuZero learns a model because, as its authors put it, planning methods "rely on knowledge of the environment's dynamics" [T1, Sec. 1]; here those dynamics are written down.

One real Form

Before the architecture, the object. This is a complete program from the public repository (`examples/surface/counter.rce` in openIE-dev/rce, at commit `1762fb4b`), fourteen lines in RCE's surface syntax, shown unedited:

```
intent "a counter that never goes negative"
    example Inc, Inc, Dec -> 1
    example Dec -> kept: Underflow

state Counter
    n: u64 = 0

event Inc
event Dec

error Underflow

step Counter on Inc -> n + 1
step Counter on Dec -> try n.checked_sub(1) else Underflow
```

Read it as $P = (S, E, \delta, D, \lambda)$. The `state` block is S, one unsigned 64-bit integer starting at zero. The two `event` lines are E. The two `step` lines are δ. The `intent` block is a small specification of intent, written $\sigma(I)$ on this page: two required traces, one of which must end in a kept step. A decrement at zero does not apply: the step keeps the state with `Underflow`, and by theorem T8 the state is unchanged. The proposer's task, stated on this example: given only the `intent` block, find the `state`, `event` and `step` lines that the gate accepts.

Architecture

The box drawn here is a proposer interface. It is not one architecture. A proposer reads the intent and the current Form and returns candidate edits or candidate Forms. The gate is outside the interface. A JEPA encoder, predicting in a latent space, is one instance. A trace-trained decoder, a diffusion code model, reinforcement learning inside a formal environment, evolutionary search, library learning and constrained decoding are other instances. Section 6 below names each one, what it needs from the checker, its known failure mode and which hypothesis it would move. The decision head is the small typed instance of the same interface. Any instance only proposes, and the Laws of Coding do not depend on it.

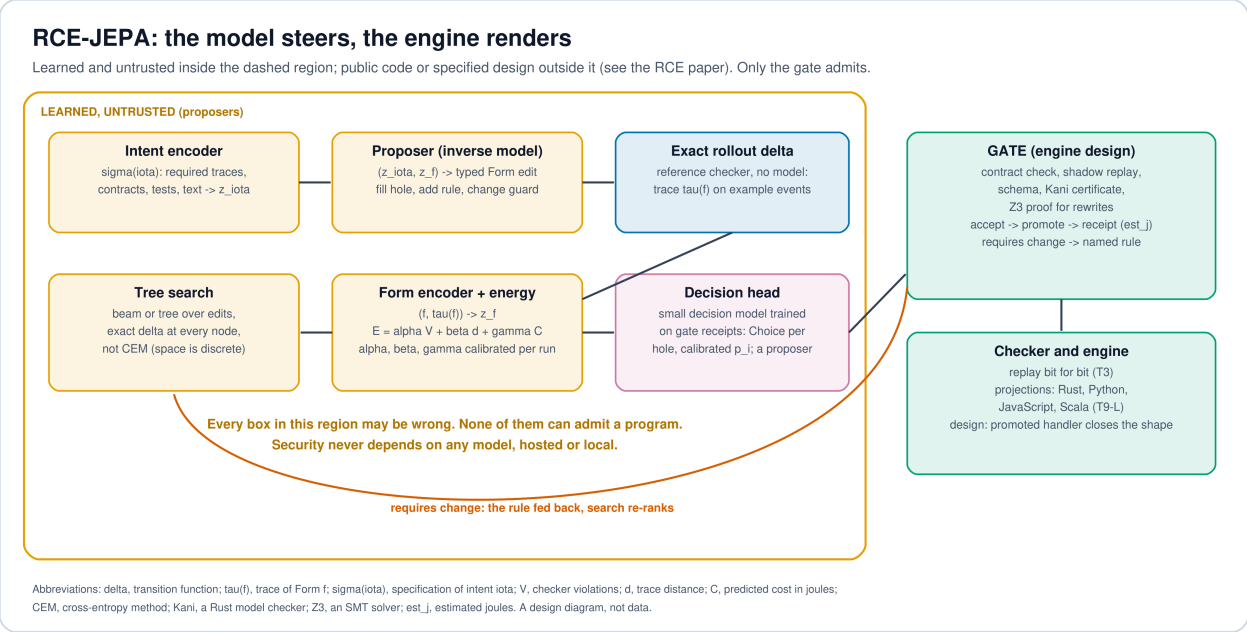

Figure 2. The model steers, the engine renders. Orange boxes are learned and untrusted; the blue box is the exact rollout by the transition function, used as a simulator during search; green boxes are the parts the RCE paper describes, public code or specified design (see that paper). Only the gate admits. A design diagram, not data. .

Component	Role	Starting design	Why
Form encoder E_f	Map a Form and its trace on example events to a latent z_f	A small transformer over the canonical JSON (JavaScript Object Notation) of the Form plus trace tokens, regularized with SIGReg (sketched isotropic Gaussian regularization) [B2]; a second arm with a discrete or soft-codebook bottleneck [B3, B4, B5] (SC-JEPA's soft codebook is stabilized together with an EMA (exponential moving average) teacher and a reconstruction anchor [B5, Sec. 4.4.1 and App. B.1])	Traces define equivalence, and the trace is free because δ is exact. Discrete tokens kept long-horizon prediction stable where continuous latents accumulated error [B3]; Boolean latents allow exact state re-identification by bitwise equality, a motivation [B4] states with reference to earlier work
Intent encoder E_I	Map $\sigma(I)$ (required traces, contracts, tests, prose) into the same space	Shared architecture, trained so that $E_I(\sigma(I))$ lies near $E_f(f^\star)$ for known accepted fits $f^\star$	Puts goal and candidate in one metric, as V-JEPA 2-AC does with goal images [B1]
Proposer Π_θ	Given z_I and the current z_f, propose a typed Form edit	A goal-conditioned inverse model, the shape LeWAM compares against as GC-IDM (goal-conditioned inverse dynamics model), a baseline from Nguyen et al. (2026) that LeWAM evaluates, not a model proposed by LeWAM's authors [B6, Sec. 4.2 and App. A.2]	With δ exact, the inverse is the learnable object
Energy $\mathcal{E}$	Rank candidates before the gate	$\alpha V(f) + \beta\, d(z_{\tau(f)}, z_{\sigma(I)}) + \gamma\, C(f)$, with $C(f)$ taken from `est_j`	Separately trained energies are not commensurate [T2, Sec. 1.3]; α, β, γ are fitted on a held-out slice to maximize AUROC (area under the receiver operating characteristic curve) for accepted against requires change, and reported with every run
Planner	Search over edits	Beam or tree search over proposer edits, scored by $\mathcal{E}$, with exact rollout at every node	V-JEPA 2-AC plans with the cross-entropy method (CEM) over continuous actions [B1]; Form edits are discrete and combinatorial, so CEM does not apply

Decision head	Bounded selection with a calibrated probability: the value for each hole, a rank, an equivalence verdict	A small decision model built by the RCE project and trained on gate receipts (next subsection)	The calibrated probability p_i orders the cheapest-first cascade

Not learned, ever: δ, λ, the fold, the hash chain, the projections, the gate and the promotion path. They are fixed and certified, and promotion never retrains them.

The decision head is a proposer, and it sits outside the gate

A class of small, energy-aware decision models appeared in 2026. A decision model takes a piece of state and a fixed question and returns a typed answer with a probability for every option, without generating text; all questions about one state are answered in a single request [B7; B8]. Answers are bounded: a yes or no, a choice among a caller-supplied set of options (up to 255 for Jev; Laya's accuracy drops beyond about 20), or a score on a declared scale [B8]. Members of the class, cited here as convergent examples of the concept and not as components of RCE:

- **Laya**, an open-weight model of about 421 million parameters built on a ModernBERT-large encoder with a two-layer decision head, released under Apache 2.0 [B8, B9].
- **Kev**, a family of open decision models from 0.8 to 27 billion parameters built on Qwen backbones, each checkpoint shipped with a fitted temperature for calibration [B10].
- **Liquid AI's open d1 models**, d1-3B and d1-omni-600M, released on 7 October 2026 for edge and physical systems: d1-3B answers one question in 16 ms on an NVIDIA Jetson AGX Thor and 50 ms on a Jetson Orin Nano, reads camera frames, and was shown navigating an environment in the Isaac Sim robot simulator with the model served on a Jetson in a hardware-in-the-loop setup [B11].

An independent benchmark of one hosted member of the class found its choice probabilities well calibrated, with an expected calibration error (ECE) of 0.028 pooled over 22 choice datasets and 279,925 answers [B7]; its binary (yes or no) probabilities were less well placed (ECE 0.052 single, 0.168 multi-label) and needed tuned thresholds [B7, Sec. 4.3]. Documented weaknesses of the class include counting, arithmetic and date comparison, and lower accuracy when the state is padded with irrelevant content [B8, B12].

The decision head is one plug in the proposer interface above. A JEPA encoder is a different plug. Hosted or local, the head only proposes.

What the RCE project plans. The project will build its own decision head, trained and calibrated on its own gate receipts. In the engine's design every gate decision writes a receipt with the candidate, the outcome, the broken rule and `est_j`, so calibration data would cost no labelling. The state handed to the head is the Form's canonical JSON and its trace, which is small by construction; that addresses the documented padded-state weakness directly. The head is a proposer. Any decision model, hosted or local, built by the project or by anyone else, only proposes; it never admits a Form, and the security of RCE never depends on it.

Data

V-JEPA 2 pairs very large passive pretraining with a small action-conditioned set [B1]. The RCE analogue has four tiers.

Tier	Source	Size target	Trains
Passive: Forms and traces	Forms lifted from pinned public corpora by the lift of the RCE paper (Sec. 6) once it is public code, Forms from the seeded generator of that paper's ecosystem protocol (Sec. 2.3), and every promoted handler	10^5 to 10^6 Forms with traces	The Form encoder, by predicting masked rules and masked trace segments in latent space
Paired: intent and fit	Tests and contracts attached to lifted programs; a Form's own trace as a self-generated specification; the preregistered brownfield applications; benchmark specifications written as Forms	10^4 pairs to start	Intent encoder and proposer
Decision receipts	Every gate decision	Unbounded, no labelling cost	The decision head's calibration

Tier	Source	Size target	Trains
Hard negatives	Requires-change verdicts with the named broken rule; model-proposed equivalences the gate did not admit (15 of 18 proposals, 12 distinct rewrites, in the public record of the model-proposer run in the RCE paper, Sec. 5.5)	Grows with every run	Contrastive signal: valid against invalid, equivalent against not

Near-duplicate Forms are removed by shape hash before any split, and held-out splits are by shape hash, never by surface text. The first corpus exists: version 1 of the Form corpus (23 October 2026) holds 192 distinct shapes of hand-written programs from the public repository (version 4, 24 October 2026), 55 of them with an intent block, and 3,456 negatives (343 conformance programs that require change, each with its diagnostic code, and 3,113 checker-verified mutants, whitespace-token and typed, that require change across 31 diagnostic codes), with a dataset card and a split by shape hash; its generator is `scripts/corpus.py` in the repository and the rows are at research.openie.dev/data/. It is the paired and hard-negative tiers at their smallest; the passive tier (lifted programs) is empty until the lift pipeline is public. On it, H1 has a first measurement, preregistered before the run: the median of rho = b(code)/b(P) over 185 programs is 9.34 against the emitted Python (interquartile range 8.45 to 10.81; version 1 gave 9.31 over 173, with 17.33 for Rust, 11.06 for JavaScript and 13.66 for Scala), above the threshold of 5; 171 of the first 173 programs were above 5 (Wilson 95% interval 0.96 to 1.00). Under a compressed description length (gzip) the ratio is 3.57 for Python and 5.73 for Rust, which is reported next to the primary so that the number is not read as a claim about information content: H1 is about the space a proposer searches, which is the token count under its vocabulary. The emitted source is the reference checker's own, with the shared runtime excluded, and no lifted or model-written program is in the population. On Apple M5 Max at commit `1762fb4b`, H2 is now measured in operating-system/on-chip `reported_j`: `rce check` takes a median of `7.32 ms` and `0.04904 J` per Form check (kernel `ri_energy_nj`) against `84.25 J` per candidate on local `Qwen 3.6 35B-A3B` (`C_check / E_model = 0.000582 <= 0.05`; details in Section 8 below). And a measure of what an intent block buys, which RJ-F2 and H7 both rest on: of the mutants that change a program's behaviour, its own example lines catch 317 of 392 (0.81, Wilson 95% interval 0.77 to 0.85; 24 of 54 programs catch every one), and the uncaught changes are almost all a constant moved by one that no example exercises.

Training, in brief

- **Encoder objective.** Latent prediction plus SIGReg, the recipe of LeJEPA, which needs a single trade-off hyperparameter and no stop-gradient or teacher-student scheme [B2, abstract], and of LeWorldModel, which trains end to end with two loss terms [B13, abstract]. Two cautions from the same line of work: the LeWorldModel authors flag the isotropic Gaussian target as a possible limitation in environments of low intrinsic dimension [B13, Sec. 4.2 and Sec. 6], and contrastive inverse dynamics is offered as a distribution-free alternative [B14], which, however, was evaluated on continuous controls only and was clearly worse than SIGReg on combinatorial puzzle tasks whose discrete state changes look alike whichever action caused them [B14, Appendix D]; and SIGReg and VISReg "do not guarantee that the learned latent representation preserves all the key information required for planning" [B15, Sec. 4.1]. Both arms run side by side, with auxiliary heads that predict exact, cheap properties of the Form (monotone tag, fuel bound, rule count).
- **Multi-step prediction.** Training with k-step recursive prediction gives significantly better results in the EB-JEPA ablations, by reducing exposure bias, with a Pareto optimum near k = 4 [B16, Fig. 3]; RoboJEPA raises the autoregressive training rollout from K = 2 to K = 10 steps in a long-horizon cooldown, which pays off only at large compute (above about 10^{21} FLOPs) [B17, Sec. 3.3 and App. C.2]; a rollout-consistency term matched to the planner horizon comes from Temporal-Distance JEPA [B18, abstract].
- **Monotone energy.** A monotone cost ranking makes plans that deviate further from a goal-reaching trajectory cost more; without it, local minima in the cost surface can lead the planner to suboptimal plans [B19]. Monotonicity is not guaranteed by the task: the EB-JEPA navigation task is non-monotonic, since the best path first moves away from the goal [B16]. Here it is a preregistered diagnostic (H11), not an assumption.
- **Proposer.** Supervised on intent, Form and edit triples from the paired tier, then improved on its own gate-accepted trajectories: the gate is the reward.
- **Never decode a latent directly.** MoRAE reports that generating motion directly in a frozen JEPA latent "fails dramatically", and traces the failure to an ill-conditioned latent spectrum and to small latent errors amplified into large artifacts after decoding [B20, abstract and Sec. 3]. The proposer outputs typed edits;

latents rank, they do not render.

Convergence: where each piece already exists

No single piece of RCE-JEPA is new. Each has an existing home, listed below as convergence, not priority. What has not been measured is the joined system, which is the subject of the hypotheses.

Piece	Where it exists	What it established	Difference from RCE-JEPA
Predict a compact plan, then decode the program	Latent Programmer [B21]	A discrete latent code predicted from the specification, then the program; string-transformation accuracy up 5 to 7 points (over 10% relative) over the best baseline, plus higher BLEU on Python code generation from docstrings [B21, Sec. 5.1 and 5.3]	Open target language, no canonical form, no checker
Learn execution, not only text	LaSynth [B22]; Code World Model (CWM) [B23]; analysis of CWMs [B24]	Latent execution improves synthesis in a restricted C domain; CWM is mid-trained on observation-action trajectories from a Python interpreter and Docker environments	Dynamics are learned; in RCE they are exact
Equivalence classes as a learning target	EqNet, Allamanis et al. [B25]	Continuous representations of symbolic expressions trained on equivalence classes	Expressions, not whole programs
Agents propose, solvers verify	Agentic model checking [B26]; verified synthesis for concept design driven by a large language model (LLM) [B27]; AlphaProof, Clover, AutoVerus (Section 3 below)	"Agents propose, solvers verify" is stated as a principle; CEGIS-style synthesis (counterexample-guided inductive synthesis) against Alloy invariants and positive and negative scenarios	Proposers generate tokens; no latent planning, no joule accounting
JEPA as a latent reasoner	JEPA-Reasoner [B28]	Latent reasoning decoupled from token generation	Text domain
Cost predicted before execution	Compiler world models [B29]	A transition model rolls out scheduling actions in latent space to predict the terminal-state representation, which a ranker uses to order candidates before fewer hardware measurements; the score is a within-workload rank, not a calibrated cost [B29, Sec. 3.2 and Sec. 6]	Tensor program schedules only
Discrete latents	Discrete-JEPA [B3]; discrete world models via regularization [B4]; SC-JEPA [B5]	Discrete tokens that keep long-horizon symbolic prediction stable [B3]; Boolean latents learned without reconstruction (planning left to future work) [B4]; a soft codebook bottleneck, with an EMA teacher and a reconstruction anchor, to stabilize one-step latent prediction, not long-horizon rollout [B5]	Synthetic images, puzzles, time series; not programs
Stable JEPA training	LeJEPA [B2]; LeWorldModel [B13]	Collapse-free training without stop-gradient, teacher networks or hyper-parameter schedulers; a standard learning-rate warm-up and cosine schedule is still used [B2, abstract and Sec. 6.1]; without stop-gradient, EMA or pretrained encoders, with two loss terms [B13, Sec. 3.1]	Reusable as an objective; validated only on images [B2] and pixels [B13], untested on program data

Piece	Where it exists	What it established	Difference from RCE-JEPA
Small calibrated decision models	Laya [B9]; Kev [B10]; Liquid AI open d1 [B11]; benchmark [B7]	Typed answers with per-option probabilities. Kev ships a fitted temperature per checkpoint [B10]; Laya ships over-confident (mean ECE 0.466) and reaches 0.081 only after a per-type temperature refit [B9]. Latency: about 33 to 40 ms per question on a T4 GPU for Laya and 18.1 ms for six questions on an H100 for Kev, but hundreds of milliseconds on a CPU or Mac (Laya 193 to 464 ms on CPU, Kev-4B 721 ms on an M5 Mac) [B9, B10]; d1-3B answers one question in 16 ms on a Jetson AGX Thor and 50 ms on a Jetson Orin Nano [B11]; one hosted member independently benchmarked for calibration [B7]	No world model; in RCE-JEPA a decision head is one proposer among several
Trace-trained proposers	Code World Model [B23], Section 3 below	Mid-training on Python execution traces and on agent trajectories. SWE-bench Verified pass@1 53.9% base and 65.8% with a best of 16 samples, as that report states	The dynamics are learned. In a Form the transition is exact. The authors call trace prediction a "first step"
Evaluator-gated evolutionary search	AlphaEvolve (Novikov et al., Section 3 below)	A model proposes edits. An automatic evaluator decides. A 4 by 4 complex matrix multiplication in 48 scalar multiplications is the kernel figure used here	Works where an evaluator exists. That report does not use a Form quotient
Constraint at sampling time	Type-constrained decoding (Mündler et al., the Swiss Federal Institute of Technology Zurich and the University of California, Berkeley, Section 3 below)	Compilation errors down 75.3% on HumanEval and 52.1% on MBPP. Syntax only: 9.0% and 4.9%	TypeScript scale. Types are not the full acceptance rule
Specification to verified code	Vericoding (Bursuc et al., Section 3 below)	12,504 specifications. Success 82.2% Dafny, 44.2% Verus, 26.8% Lean. The abstract rounds those shares to 82%, 44% and 27%	The specification still has to be written. Natural-language descriptions add little

The specification cost, named

The checker guarantees that a Form is valid; it does not guarantee that it is the Form that was meant. In verified LLM-driven synthesis for concept design, invariant-only synthesis "reached verified designs quickly but often produced inconsistent designs across runs", which shows "that invariants alone underconstrain the design task"; scenario-guided synthesis recovered intended designs more consistently than natural-language prompting, although minimal scenarios could overfit [B27, abstract], on 3 applications and 12 variants with one LLM configuration [B27, Sec. 5]. The dominant failures "were not failures of bounded verification, but failures of specification coverage" [B27, Sec. 5.6]. In seL4 the abstract specification took about 4 person-months and the proof about 20 person-years (Klein et al. 2009, Section 2 below). A closed program world makes acceptance exact and search tractable. It moves the uncertainty of intent into $\sigma(I)$; it does not remove it. Falsifier RJ-F2 below makes that cost public and measurable.

Proposer hypotheses

The proposer is tested by H11 (monotone energy), H12 (decision-head calibration) and H13 (discrete against continuous latents), and is scored on the residual of the deterministic ladder above (H14, H15). All fifteen hypotheses, H1 to H15, are stated in numerical order with their thresholds in one table in Section 8 below, under the same conventions: preregistered before the run, frozen engine commit and binary hash recorded, same gate for every arm.

The H3 control is fixed now: the same gate and the same cost accounting, with Qwen2.5-Coder-1.5B proposing surface code, the arm already run in the RCE paper (Sec. 5.5), where the gate did not admit 15 of 18 proposed equivalences and admitted only one distinct rewrite, a rule the engine already had.

Falsifiers. The falsifiers for the premise itself, RJ-F1 to RJ-F6, now follow the hypotheses they test, directly after the H1 to H15 table in Section 8 below.

Risks and mitigations

Risk	Evidence	Mitigation
Decoding a latent amplifies small errors	Direct generation in a frozen JEPA latent fails dramatically for motion, from spectral ill-conditioning and decoder amplification [B20]	Never decode a latent; the proposer emits typed edits and the gate checks them
A representation that does not collapse but learns the wrong thing	"Physical representation laziness" under SIGReg [B15]	Auxiliary exact-property heads; an inverse-dynamics regularizer
Isotropic prior misfits a low-dimensional Form space	[B13]; [B14]	SIGReg and contrastive inverse-dynamics arms side by side, remembering that the inverse-dynamics arm was weaker on discrete puzzle state [B14]
Energy terms not commensurate	[T2, Sec. 1.3]	Fit α, β, γ per experiment and publish them
Decision head degrades on padded state	Documented weakness of the class [B8, B12]	Feed only canonical Form JSON and trace
Expressiveness ceiling	Rice's theorem [T3]; the restricted language	Lift coverage is the measurement; RJ-F1 is the falsifier
Specification cost dominates	seL4 effort (Section 2 below); invariant under-constraint [B27]	RJ-F2; tools that capture traces from running programs and infer contracts, as deliverables in their own right
Benchmark oracle weakness	PatchDiff (Wang, Pradel and Liu, Section 3 below): 7.8% of patches counted correct fail developer tests, and resolve rates are inflated by about 6.4 points. A METR trial of early-2025 tools (Becker et al., Section 3 below): 16 developers, 246 tasks, measured completion time 19% higher. The forecast was 24% lower. A post-hoc estimate was 20% lower	Acceptance is the gate plus replay, not a test suite. Section 8 (H15) enforces `rce check` plus obligation preservation on every candidate
Learned planning latency	V-JEPA 2 model predictive control took about 16 seconds per action [B1]	Search over Forms with an exact checker (`7.32 ms` / `49.04 mJ` on Apple M5 Max) and Gear-0 crystallization (`0.325 µs` / `1.77 µJ` per lookup) bound repeated rollouts

The phased pilot plan, from the data engine through convergence, with preregistration and exit criteria for each phase, is kept with the project's working notes and summarized here: Phase 0 builds the lift corpus and receipt dataset, and measures the deterministic ladder for H14, which needs no learned proposer; Phase 1 trains the Form encoder and tests H1, H5 and H11; Phase 2 builds and calibrates the decision head (H12); Phase 3 trains the dedicated JEPA proposer and planner against the token control (H3, H4, H10, H13), extending the first single-shot off-the-shelf H15 baseline reported in Section 8; Phase 4 extends on-chip `reported_j` (H2) to external wall-meter `measured_j` (H8, H9); Phase 5 runs the convergence protocol, in which Gear-0 promotion takes the model out of the loop for each closed shape. Alongside Phase 0, a slice of the vericoding benchmark (source C34, listed under Convergence by commitment below), chosen as its state-machine-shaped tasks in Dafny, Verus and Lean, is lifted into Forms and scored under RCE's gate next to the published vericoding results, an outside benchmark for the step from specification to Form (H6, H7 and H8).

Proposer references

References marked [T] are listed under Thesis references below; references to Sections 2, 3 and 5 are to the sections of this page below. Labels follow the convention of the main references below. **Full** here means the whole text was read, including references and appendices; for B9 and B10 the live model card and README were read on 14 October 2026.

1. [B1] M. Assran et al., "V-JEPA 2: Self-Supervised Video Models Enable Understanding, Prediction and Planning", 2025. arXiv:2506.09985. Full.
2. [B2] R. Balestriero and Y. LeCun, "LeJEPA: Provable and Scalable Self-Supervised Learning Without the Heuristics", 2025. arXiv:2511.08544. Full.

3. [B3] J. Baek et al., “Discrete JEPA: Learning Discrete Token Representations without Reconstruction”, 2025. arXiv:2506.14373. Full.
4. [B4] D. Bizzaro and L. Serafini, “Discrete World Models via Regularization”, 2026. arXiv:2603.01748. Full.
5. [B5] Y. He et al., “SC-JEPA: Stabilizing Latent Predictive Learning for Time-Series Anomaly Prediction”, 2026. arXiv:2602.04643. Full.
6. [B6] M. Fu et al., “LeWAM: End-to-End World and Action Modeling with JEPAs”, 2026. arXiv:2610.12818. Full.
7. [B7] T. Deußer, L. Sparrenberg and R. Sifa, “Evaluating and Benchmarking the System One Model Jev”, 2026. arXiv:2609.37647. Full.
8. [B8] W. Wu, “Jev vs Laya: Comparing and Choosing Between Closed-Source and Open-Source System One Decision Models”, 23 September 2026. wilsonwu.me. Full.
9. [B9] Nandakishor M (Convai Innovations), Laya model card. Hugging Face. Full (read live 14 October 2026).
10. [B10] J. Palmer, Kev. GitHub. Full (README, read live 14 October 2026).
11. [B11] Liquid AI, “Open d1: Edge decision models for text, vision, and audio”, 7 October 2026. liquid.ai. Full.
12. [B12] Sanity, “Decision model” (glossary). sanity.io. Full.
13. [B13] L. Maes, Q. Le Lidec, D. Scieur, Y. LeCun and R. Balestriero, “LeWorldModel: Stable End-to-End Joint-Embedding Predictive Architecture from Pixels”, 2026. arXiv:2603.19312. Full.
14. [B14] J. Boylan and C. Hokamp, “No Gaussian Required: Contrastive Inverse Dynamics for JEPA World Models”, 2026. arXiv:2608.17542. Full.
15. [B15] P. Zhu et al., “Spectral-Target Physical Latent Structuring for JEPA-Style World Models”, 2026. arXiv:2609.04264. Full.
16. [B16] B. Terver et al., “A Lightweight Library for Energy-Based Joint-Embedding Predictive Architectures”, 2026. arXiv:2602.03604. Full.
17. [B17] A. Zholus et al., “RoboJEPA: Scaling Robotic Latent World Models”, 2026. arXiv:2610.10515. Full.
18. [B18] J. Bai and J. Xiong, “Temporal-Distance JEPA: Plan-Aware Representation Learning for Latent World Model Predictive Control”, 2026. arXiv:2607.25337. Full.
19. [B19] A. Chahe, S. Cai and L. Zhou, “Latent World Models with Monotone Planning Costs for Image-Goal Navigation”, 2026. arXiv:2608.09073. Full.
20. [B20] Y. Zhu et al., “MoRAE: Flow-Friendly Self-Supervised Latents for Text-to-Motion Generation”, 2026. arXiv:2607.29180. Full.
21. [B21] J. Hong, D. Dohan, R. Singh, C. Sutton and M. Zaheer, “Latent Programmer: Discrete Latent Codes for Program Synthesis”, ICML 2021. PMLR PDF. Full.
22. [B22] X. Chen, D. Song and Y. Tian, “Latent Execution for Neural Program Synthesis”, NeurIPS 2021. arXiv:2107.00101. Full.
23. [B23] Meta FAIR CodeGen Team, “CWM: An Open-Weights LLM for Research on Code Generation with World Models”, 2025. arXiv:2510.02387. Full.
24. [B24] B. Rahmani, “Debugging Code World Models”, 2026. arXiv:2602.07672. Full.
25. [B25] M. Allamanis, P. Chanthirasegaran, P. Kohli and C. Sutton, “Learning Continuous Semantic Representations of Symbolic Expressions”, ICML 2017. arXiv:1611.01423. Full.
26. [B26] Y. Sun, J. Liu, D. Kroening and J. Xue, “Agentic Model Checking”, 2026. arXiv:2605.21434. Full.
27. [B27] A. Cunha, “Verified LLM-Driven Synthesis for Concept Design”, 2026. arXiv:2607.15718. Full.
28. [B28] B. K. Liu, Z. P. Chen and D. P. Woodruff, “JEPA-Reasoner: Decoupling Latent Reasoning from Token Generation”, 2025. arXiv:2512.19171. Full.
29. [B29] H. Pan et al., “Toward Compiler World Models: Learning Latent Dynamics for Efficient Tensor Program Search”, 2026. arXiv:2606.09312. Full.
30. [B30] Open Interface Engineering, “Exact-proposer ladder: registered run”, openIE-dev/openie-research-experiments, `rce/bench/ladder/exact-proposers/`: registration committed before the run at `3030469d`, results at `70a0a776` (pull request #21); ladder openIE-dev/rce `6d04a096`. GitHub. Full.
31. [B31] Open Interface Engineering, “Exact-proposer ladder on real defects: registered run”, openIE-dev/openie-research-experiments, `rce/bench/ladder/real-defects/`: registration committed before the run at `64e70389`, results at `00e3b005` (pull request #22); ladder openIE-dev/rce `7dc4eded`. GitHub. Full.
32. [B32] Open Interface Engineering, “Exact-proposer ladder with renames: registered run”, openIE-dev/openie-research-experiments, `rce/bench/ladder/rung-e/`: registration committed before the run at `be5f370d`, results at `3a5227c2`, merged at `f76ba9d8` (pull request #23); ladder openIE-dev/rce `8a5d90ea`. GitHub. Full.

33. [B33] Open Interface Engineering, "RCE-JEPA H2, single-shot H15, and Shape D (DNULLMM1) Gear-0 on the 40 open residual tasks", registered 22 November 2026 (`rce-jepa-h15-and-shape-d-gear0-2026-11-22`, `sha256: b64c4c671ae85755...`) on `openIE-dev/rce` at commit `1762fb4b` (20 November 2026, binary `sha256: 1d3b0d07ff5b2e8f...`), Apple M5 Max dual-channel `reported_j` receipts (`measured_j = None`, `board_metered = false`). research.openie.dev/data/. Full.
34. [B34] Open Interface Engineering, "FormJEPA-v1 latent Form repair proposer, 3-Gear Cascade, and rce repair rung f criterion evaluation on the 151-task ladder and 40 residual tasks", registered 22 November 2026 (`rce-jepa-form-jepa-v1-2026-11-22`, `sha256: 22682e6e936ec120...`; `rce-repair-rung-f-wave3-eval-2026-11-22`, `sha256: 04aff4b2b44cdbea...`) on `openIE-dev/rce` at commit `1762fb4b` (Linux x86_64 binary `sha256: 5ef9931ff23a310f...`), AMD Ryzen AI Max+ 395 Linux powercap RAPL `reported_j` receipts (`measured_j = None`, `board_metered = false`). research.openie.dev/data/. Full.
35. [B35] Open Interface Engineering, "FormEBM-v2 contrastive multi-step margin latent energy scorer (H5, H11) and H15-Iterative bounded checker-feedback residual closure (40/40) with DNULLMM1 Gear-0 v2 crystallization", registered 23 November 2026 (`rce-jepa-form-ebm-v2-2026-11-23`, `sha256: 128f17e69361ea89...`; `rce-jepa-h15-iterative-2026-11-23`, `sha256: a7774d3bd03ae450...`) on `openIE-dev/rce` at commit `1762fb4b` (Linux x86_64 binary `sha256: 331464496009d98a...`), AMD Ryzen AI Max+ 395 / Radeon 8060S Linux powercap RAPL `reported_j` receipts (`measured_j = None`, `board_metered = false`). research.openie.dev/living/rce-jepa/ · research.openie.dev/data/. Full.
36. [B36] Open Interface Engineering, "FormEBM-v3 span-normalized two-head decomposed contrastive energy scorer (H5, H11, and zero-shot OOD H15 cross-span transfer)", registered 23 November 2026 (`rce-jepa-form-ebm-v3-2026-11-23`, `sha256: cb7ee1ac8dc5c594...`) on `openIE-dev/rce` at commit `1762fb4b` (Linux x86_64 binary `sha256: 386be0c26389550e...`; checkpoint `sha256: 71e642f1bb0b9c06...`), AMD Ryzen AI Max+ 395 / Radeon 8060S Linux powercap RAPL `reported_j` receipts (`measured_j = None`, `board_metered = false`). research.openie.dev/living/rce-jepa/ · research.openie.dev/data/. Full.

The thesis: software construction is pre-"Laws of Coding"

Picture Europe in the 1660s. Kepler had published two regularities of planetary motion in *Astronomia nova* (1609) and a third in *Harmonice mundi* (1619) [T4]. Galileo and Huygens had derived testable conclusions from hypotheses about motion [T5, Sec. 1]. Gunners, millwrights and shipwrights built by rule of thumb and long apprenticeship. The sky had been watched with enormous care for two thousand years, and yet prediction was a patchwork. Newton, by the account of George Smith, "was familiar with seven different approaches to calculating planetary orbits, all at roughly the same accuracy", and calculated positions "were sometimes off by as much as a fourth of the width of the Moon" [T5, Sec. 2]. There were many good tables and no single account of why the tables worked.

Then, in 1687, came the *Principia*. A handful of axioms of motion and one inverse-square force, and from them trajectories that could be computed in advance, costs that could be reckoned before the cannon was cast, and errors that pointed to new bodies rather than to bad luck. Smith summarizes the change: after the *Principia*, "physical theory gained primacy over observation for purposes of answering specific questions about the world" [T5, Sec. 1]. Galileo's results for uniform gravity did not disappear; they became a limit case of inverse-square gravity, and the theory went on to explain the planets' deviations from Kepler's rules [T5, Secs. 1 and 2].

The premise, in plain words: **the practice of software construction, including today's AI-assisted coding by large language models, is pre-"Laws of Coding".** Here "the practice of software construction" means the practitioners and systems, human developers and code-generating models alike, that produce code by empirical trial, testing and review rather than by derivation from a formal rule set. That practice today lives in the 1660s. It has careful observation: pass@k (the chance that at least one of k sampled programs passes the tests), used in Section 4 below, and SWE-bench Verified (a reviewed set of repository issues), used in Section 3 below. It has craft: prompt wording, retries, and trial and error. Outputs of code-generating models are samples from a probability distribution, so the same request can give a different program tomorrow; when ChatGPT (gpt-3.5-turbo at its default temperature) was asked the same coding question five times, for 47.56% to 75.76% of problems across three datasets there was no test on which all five answers gave the same output, and "setting the temperature to 0 does not guarantee determinism in code generation" (Ouyang et al., 2023, abstract). The regularities that the human side of the practice does show are, in Lehman's words, ones that "arise from the habits and practices of people and organizations" [T6, Sec. V-A, p. 1069]. Hoare and colleagues named the same gap in 1987: "What then are the laws of programming, which provide the formal basis for the profession of software engineering? Many programmers may be unable to quote even a single law" [T7, p. 673]. Dijkstra had already marked the limit of the empirical method

that the practice still relies on: "program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence" [T8, Argument three]. There is no agreed conserved quantity for a program, no equation of motion that a code generator obeys, and cost is measured after the fact, in tokens and in electricity, rather than computed before the work begins.

"Laws of Coding" names the craft layer, defined with the other three layers below. The RCE-JEPA proposal, stated here as conjecture, is that this craft can gain a small set of rules from which the behaviour and the cost of the code it produces can be computed before it runs, by borrowing from the layers beneath it. The candidates are (L1) a deterministic transition rule, (L2) an account of information and energy per transition, (L3) a canonical Form for every program, which is compression, and (L4) a composition rule under which L1 to L3 survive when programs are joined. The Reality Code Engine (RCE) proposes candidates for each; hypotheses H1 to H15 are the tests, stated in numerical order in one table in Section 8 below. Nothing in this section is a measured result unless it carries a number and a source.

The analogy has limits, stated here. Newton described a world nobody designed; programs are made by people, so their rules are partly chosen rather than discovered. Mechanics has continuous quantities and exact conservation of energy; a program's state is discrete and finite, and the energy a chip spends depends on the hardware, not only on the program. Rice's theorem [T3] says that no general procedure decides any non-trivial property of what an arbitrary program computes, and the halting problem [T9] is the best-known case; RCE avoids these limits only by working in a restricted language whose programs carry termination bounds, and that restriction has a price in what can be expressed. Finally, even the *Principia* was not the "clockwork universe": Smith notes that aspect "was added by Laplace" [T5, Sec. 1]; the axioms of motion alone did not settle every problem of mechanics either, since for bodies that are not point-masses they "must be supplemented by further principles" [T5, Sec. 5]; and "Newtonian theory is now seen to hold only to high approximation in limited circumstances" [T5, Sec. 1]. The claim is an analogy, not an equivalence.

Four layers of law, from theory to art

Four phrases are easy to blur, and this page keeps them apart. Each names a different layer, running from theory through science to art, and each is formalized to a different degree today. They are never used interchangeably here.

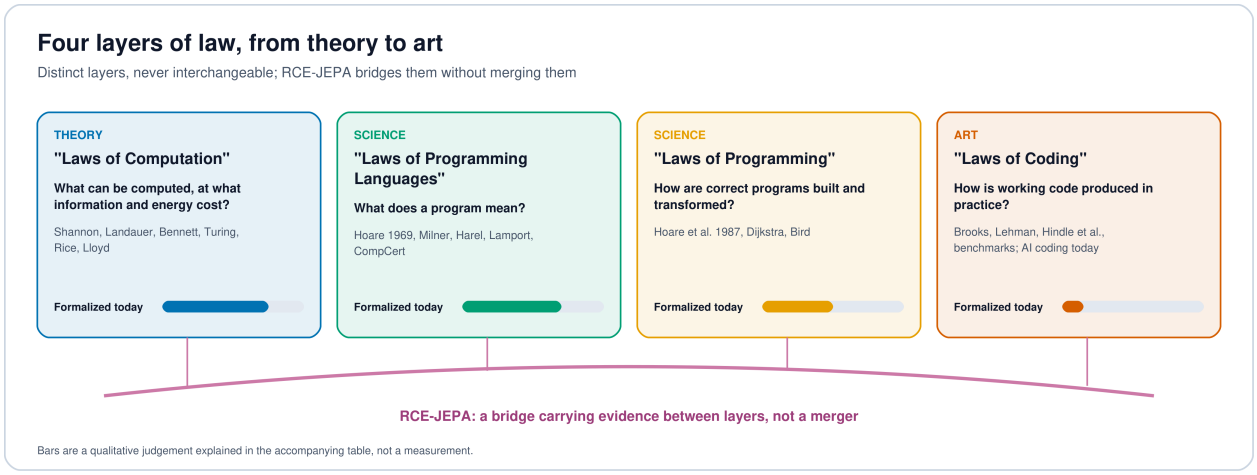

Figure 3. The spectrum. Left to right: "Laws of Computation" (theory), "Laws of Programming Languages" and "Laws of Programming" (science), "Laws of Coding" (art). The bar under each layer shows how formalized it is today, a qualitative judgement explained in the table below. RCE-JEPA is drawn as a bridge that carries evidence between layers; it does not merge them. A qualitative diagram, not data. .

Layer	Definition	Who works on it	How formalized today
"Laws of Computation"	Physical and information limits: what can be computed at all, and at what cost in information, time and energy.	Shannon [T10]; Landauer [T11]; Bennett and Landauer [T12, T13]; Turing [T9] and Rice [T3]; Lloyd [T14]; Wolpert [T15]; Carette, James and Sabry [T16].	Exact theorems for limits: information in bits, the erasure floor $k_B T \ln 2$, undecidability. No unified account yet predicts the cost of a given program on a given machine; Bennett and Landauer note their limits are "terribly far away from the real limits of modern technology" [T13, p. 48].
"Laws of Programming Languages"	Semantics and type theory: the rules that fix what a program in a language means.	Hoare's axiomatic semantics [T17]; Milner [T18]; Harel [T19]; Lamport [T20]; Berry and Gonthier [T21]; Halbwachs et al. [T22]; Leroy's verified compiler and Letouzey's extraction (Section 1 below).	Fully formal for some languages and compilers, machine-checked in CompCert (Section 1 below). Formal meaning is a property a language is designed to have, not one every language has.
"Laws of Programming"	Algebraic laws of correct program construction: equations by which programs are transformed and derived from specifications.	Hoare, Hayes, He, Morgan, Roscoe, Sanders, Sorensen, Spivey and Sufrin [T7]; Dijkstra's derivation of programs and Bird's calculus of lists (Section 1 below).	Nearly one hundred laws for a small language [T7]; by its authors' own account "still a long way from knowing how to apply them directly to the design of correct and efficient programs on the scale required by modern technology" [T7, p. 686].
"Laws of Coding"	The empirical craft of producing working code by people or models. This is where AI coding sits today.	Brooks [T23]; Lehman [T6]; Hindle et al. [T24]; the reports in Section 3 below.	Empirical regularities and benchmarks, no laws. Outputs of AI coders are samples, so the same request can give a different program (Ouyang et al., 2023, quoted above).

How RCE-JEPA relates to the layers. RCE-JEPA is a bridge, not a merger. It works in the "Laws of Coding" layer, where a learned model proposes code, and it ties each proposal to the layers beneath: the Form has a fixed meaning ("Laws of Programming Languages"), accepted Forms can be transformed by stated rules ("Laws of Programming"), and their cost is to be accounted against physical limits ("Laws of Computation"); RCE counts that cost today in integer units, which are not joules. Each hypothesis below names the layer it tests. If the bridge holds, the craft gains rules borrowed from the science and the theory; the layers stay distinct.

Mechanics has its *Principia*. Computation does not, yet. The pieces of "Laws of Computation" listed above are exact, but they have not been joined into one formal account that predicts behaviour and cost together, the way the *Principia* joined Kepler's and Galileo's regularities. This is stated as an open problem, not a solved one.

Nothing in that gap is a failure of the people who came before. Each era did what its machines, its theory of information and its craft of building and optimizing programs allowed. The timeline below says what each era had to work with, with figures from the sources named, and what it achieved.

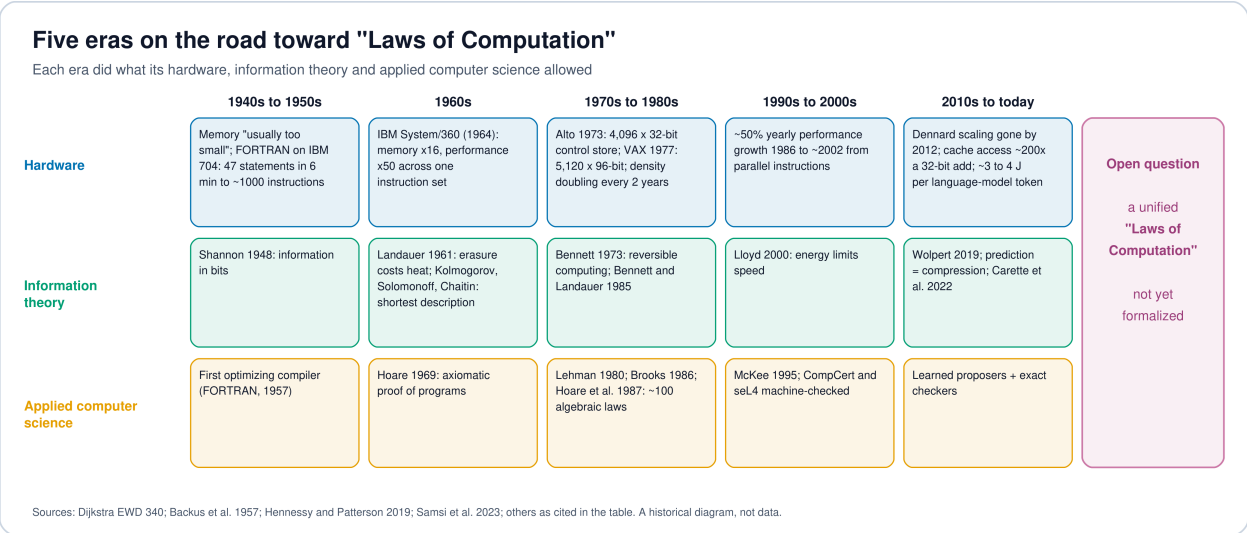

Figure 4. Five eras on the road toward "Laws of Computation". Top lane: hardware, with figures from the sources in the table below. Middle lane: information theory. Bottom lane: applied computer science for developing and optimizing algorithms (compilers, complexity, synthesis, verification). The right-hand box is the open question: a unified "Laws of Computation". A historical diagram, not data. .

Era	Hardware (memory, operations, energy)	Information theory	Applied computer science for algorithms	What it made possible, and where it stopped
1940s to 1950s	Single-copy machines whose memory "was usually too small" and which were "usually too slow" [T8]. On the IBM 704, a 47-statement FORTRAN (FORmula TRANslation) program compiled "in six minutes, producing about 1000 instructions" [T25, PDF p. 2].	Shannon (1948) makes information a measurable quantity: "The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point" [T10, Introduction, p. 1].	The first optimizing compilers [T25].	Translation from formula to machine code became automatic. Every instruction and every word of memory was dear, so effort went to squeezing programs into machines, as Dijkstra recalls [T8].
1960s	The four models of IBM System/360 (1964) differed by a factor of 16 in memory capacity and 50 in performance while sharing one instruction set [T26].	Landauer (1961) ties erasure of information to heat [T11]; Solomonoff, Kolmogorov and Chaitin define complexity as shortest description [T27, T28, T29].	Hoare's axiomatic basis for proving programs correct (1969) [T17].	Correctness and information got exact definitions on paper. Dijkstra (1972): as "the power of available machines grew by a factor of more than a thousand, society's ambition to apply these machines grew in proportion" [T8], and the software crisis followed.
1970s to 1980s	Alto (1973): a writable control store of 4,096 words of 32 bits; VAX-11/780 (1977): a control store of 5,120 words of 96 bits; Moore's 1975 projection of a doubling in transistor density every two years [T26].	Bennett (1973) shows that general-purpose computing machines "may be made logically reversible at every step, while retaining their simplicity and their ability to do general computations" [T30, abstract, p. 525]; Bennett and Landauer (1985) set out the physical limits [T13].	Brooks (1986) separates essential from accidental complexity [T23]; Hoare and eight co-authors give nearly one hundred algebraic laws for a programming language [T7]; Lehman states empirical laws of software evolution [T6].	Programs became algebraic objects. Hoare et al. are candid about the distance left: "we are still a long way from knowing how to apply them directly to the design of correct and efficient programs on the scale required by modern technology" [T7, p. 686].
1990s to 2000s	Instruction-level parallelism and faster transistors gave about 50% annual performance growth from 1986 to about 2002 [T26].	Lloyd (2000) bounds operations per second by energy [T14].	McKee (1995) proposes fundamental laws of computation [T31]; machine-checked compilers and kernels, CompCert and seL4 (Section 1 below).	Proof reached real systems, at a price: seL4's proof took about 20 person-years (Section 1 below).
2010s to today	Dennard scaling "faded to almost nothing by 2012"; a 32-kilobyte cache access costs about 200 times the energy of a 32-bit add, and off-chip DRAM (dynamic random-access memory) about 10 times a last-level cache access; Google's TPU (Tensor Processing Unit) v1 performs 256 x 256 multiply-accumulates per cycle with 24 megabytes of local memory, at more than 80 times a CPU's energy efficiency on its workload [T26]. A 65-billion-parameter language model used about 3 to 4 joules per output token (Samsi et al., Section 4 below).	Wolpert's stochastic thermodynamics of computation [T15]; prediction shown equivalent to compression for language models [T32]; Carette, James and Sabry rebuild models of computation on conservation of information [T16].	Learned code generators; model plus verifier loops; deterministic simulation testing (see who holds these views, below).	Checking is cheap and exact where proposers are cheap and plentiful. A Python matrix multiply rewritten for the hardware ran "more than 62,000x faster" [T26]: the distance between a program and its machine is now large, measurable and worth closing.

Why progress is possible now (opinion, to be tested). Three things that were never available together are

available today. Machine-checked verifiers exist for real systems, so a rule can be enforced rather than hoped for. Learned models can propose candidate programs by the thousand, so the search for a program that satisfies a rule is affordable. And energy can be read from counters on ordinary hardware, so cost can be measured against prediction rather than guessed. Each earlier era lacked at least one of the three. None of this makes "Laws of Computation" inevitable; it makes them testable, which is the hypotheses' purpose below.

Symbols used in this section

- $P = (S, E, \delta, D, \lambda)$: a program as a functional state machine. S is a finite set of states; E a finite set of events (inputs); $\delta : S \times E \to S$ the transition function; D the set of derivations, values computed from a state by pure functions; $\lambda : S \times E \to O$ the output function into a finite output set O.
- Ω_n: the set of canonical Forms of size at most n symbols. A Form is the checker-accepted canonical representation of a program in RCE (Section 8).
- X: the set of surface programs (the text people or models write). $\kappa : X \to \Omega$: the canonicalization map from surface program to Form.
- $H(\cdot)$: Shannon entropy in bits [T10]. $K(x)$: Kolmogorov complexity, the length in bits of the shortest program that outputs x on a fixed universal machine [T27, T28].
- $k_B = 1.380649 \times 10^{-23}\ \mathrm{J\,K^{-1}}$: Boltzmann's constant. T: temperature in kelvin (K). J: joule.
- JEPA: Joint Embedding Predictive Architecture [LeCun 2022, cited above]. MDL: minimum description length [T33]. EBM: energy-based model.

Part 1. Computation as a finite information system

Strip a program of its syntax, its comments, its indentation and its author's habits, and what remains is a machine that moves from one state to the next, one event at a time. That machine is finite, and a finite thing can be counted.

Definition. A program is a deterministic functional state machine $P = (S, E, \delta, D, \lambda)$ with S and E finite and δ a total function. Determinism means δ is a function, not a relation: one state and one event give exactly one next state.

Bound on information content. If a Form is drawn from Ω_n by any distribution, its entropy is at most that of the uniform distribution: for n outcomes, "H is a maximum and equal to log n when all the $\mathrm{p_i}$ are equal" [T10, Sec. 6, property 2, p. 11]:

$$H(P) \ \le \ \log_2 |\Omega_n| \text{ bits.}$$

For an alphabet of a symbols, $|\Omega_n| \le \sum_{k \le n} a^k$, so $H(P) \le n \log_2 a + 1$ bits. Every Form, however long its behaviour, is a finite message.

Compression and understanding. Kolmogorov complexity $K(x)$ [T27, T28] measures the shortest description of an object. Kolmogorov takes as the "relative complexity" of an object y given x "the minimal length l(p) of the 'program' p for obtaining y from x" [T28, Sec. 3, p. 165]; Solomonoff's induction [T27] and Rissanen's MDL [T33] turn "prefer the shortest description" into a rule for prediction and model choice. The bridge to learning is Shannon's source coding theorem [T10, Sec. 9, Theorem 9, p. 16]: a good code spends about $-\log_2 p(x)$ bits on x (Shannon's second proof gives a message of probability p_s a code of m_s binary digits with $\log_2(1/p_s) \le m_s < 1 + \log_2(1/p_s)$ [T10, Sec. 9, p. 17]), so a better predictor is a better compressor. Delétang et al. state it directly: "maximizing the log2-likelihood (of the data) is equivalent to minimizing the number of bits required per message" [T32, Sec. 1]. K itself is not computable: Kolmogorov already noted that the conditional complexity "cannot be effectively calculated (generally recursive) even if it is known to be finite for all x and y" [T28, Sec. 3, p. 165], and Schmidhuber notes "there is no general way of finding the shortest program computing any given data" [T34, Sec. 1]. In a restricted language with a canonical Form, the Form length is a computable upper bound on K up to an additive constant.

Part 2. Compression into Form and template

Ask a hundred programmers to write a shopping cart and you will get a hundred programs. Underneath the hundred there are perhaps three or four ideas. The surface is a crowd; the Form is the face it shares.

Equivalence classes. Define $x \sim y \iff \kappa(x) = \kappa(y)$. Each Form $f \in \Omega$ is the representative of the class $[f] = \kappa^{-1}(f)$ of $m_f = |[f]|$ surface programs. Because κ is a function, the chain rule of entropy, $H(x, y) = H(x) + H_x(y)$ [T10, Sec. 6, property 5, p. 12], gives

$$H(X) \;=\; H(\kappa(X)) \;+\; H\big(X \mid \kappa(X)\big).$$

A predictor that outputs the Form rather than the surface text is spared the second term. If surface variants within a class are equally likely, the saving per Form is

$$\Delta H_f = \log_2 m_f \text{ bits}, \qquad \rho_f = \frac{|x|}{|f|} \quad \text{(bytes of surface code per byte of Form)}.$$

This is why the predictor works on the quotient space $X/\!\sim \;\cong\; \Omega$: it learns a smaller, cleaner target, and every point in it is something the checker can judge.

Templates as low-dimensional manifolds. A template or a no-code schema is a Form with holes. If a template has h holes each taking one of v values, it covers v^h Forms and is addressed by $h \log_2 v$ bits, typically far fewer than $\log_2|\Omega_n|$. Program space is vast; the programs people actually want lie on thin sheets within it. This is a conjecture, consistent with the observation that code is more predictable than natural language [T24] and with learned code embeddings in which "semantically similar objects are mapped to close vectors" [T35, Sec. 1].

Corpus measurements. H1 has a published floor on the hand-written population. The median of $\rho =$ b(code)/b(P) over 185 programs is 9.34 against the emitted Python, above the threshold of 5. The run was registered before it was measured. The data section of the proposer above gives the interquartile range, the version-1 figure of 9.31 over 173 programs, and the compressed-description check. The saving ΔH on a lift corpus is still planned. That corpus is empty until the lift pipeline is public. The 9.34 figure is a measured floor on hand-written programs. It is not a measurement of lifted or model-written programs.

Part 3. Kinematics: a Newtonian form of computing

Throw a stone and, given where it starts, how fast, and what pushes on it, Newton tells you where it lands. A deterministic program has the same character, and more strictly: no air, no wind, no rounding of the world. Given where it starts and what events arrive, its entire future is fixed.

Trajectory. For initial state s_0 and event stream $e_0, e_1, \ldots, e_{T-1}$,

$$s_{t+1} = \delta(s_t, e_t), \qquad t = 0, \ldots, T-1.$$

The correspondence proposed here, as analogy:

- state s_t plays the part of position;
- an event e_t plays the part of an applied impulse;
- δ plays the part of the equation of motion;
- the event log is a record from which the trajectory can be replayed exactly, which is reversibility in the record even when δ discards information; Bennett shows that "it is possible to reprogram any deterministic computation as a sequence of logically reversible steps, provided the computation is allowed to save a copy of its input" [T12];
- step count or fuel t plays the part of time;
- energy per transition, in joules, is the cost.

Energy floor. Landauer argued that logical irreversibility "is associated with physical irreversibility and requires a minimal heat generation, per machine cycle, typically of the order of kT for each irreversible function" [T11, abstract, p. 183]; resetting a bit cuts the entropy by k $\log_e 2$ and so supplies "0.6931 kT per restored bit to the surroundings" [T11, Sec. 4, p. 187]. Landauer's principle, as Bennett later restates it, holds that a logically irreversible operation such as erasure "must be accompanied by a corresponding entropy increase" in the environment [T12, abstract]. The floor per erased bit is

$$E_{\min} = k_B T \ln 2 \approx 2.87 \times 10^{-21}\ \text{J} \quad \text{at } T = 300\ \text{K}.$$

If transition t erases b_t bits, then $E_{\text{total}} \geq k_B T \ln 2 \sum_t b_t$. Lloyd adds an upper limit on speed: "the speed with which a physical device can process information is limited by its energy" [T14, abstract], at most $2E/\pi\hbar$ operations per second for energy E, where $\hbar = 1.0546 \times 10^{-34}$ J s is the reduced Planck constant. Real chips spend many orders of magnitude above the Landauer floor per operation; the RCE project has no measured joules-per-operation figure at the time of writing, and H9 (Section 8 below) is the planned measurement. Kolmogorov flagged the missing cost dimension in 1965: his complexity measures "do not allow for the 'difficulty' of preparing a program p for passing from an object x to an object y" [T28, Sec. 4, p. 167]. Wolpert, quoting Boyd, notes as a caution that "different implementations of the same computation (input to output) can have radically different thermodynamic consequences" [T15, Sec. I.B]: energy belongs to the program and the machine together.

Definition (kinematic program). P is kinematic if, for every s_0 and every finite event stream,

$$\text{(i)}\ \ \mathrm{Var}[s_T] = 0, \qquad \text{(ii)}\ E_{\text{total}} = \sum_{t=0}^{T-1} c(s_t, e_t)\ \leq\ c_{\max} F$$

where $c : S \times E \to \mathbb{R}_{\geq 0}$ is a cost table in joules for a stated machine, $c_{\max}$ its largest entry, and F the fuel bound (maximum steps) from the termination proof. Condition (i) says the trajectory has zero variance; condition (ii) says its cost can be computed, or at least bounded, before it runs. RCE's own cost table, in Appendix C.6.4 of its specification, is in integer units, not joules; a table in joules per unit for a stated machine is a proposal in the specification repository, and H9 is the test.

Contrast. A stochastic generator returns a sample $x \sim p_\theta(x \mid \text{prompt})$ from a model with parameters θ. Its output is a distribution, its correctness is a probability, and its cost is known only after sampling. RCE-JEPA keeps the generator stochastic but makes the thing it generates kinematic, and puts a deterministic checker between them.

Part 4. A world model whose physics is known exactly

Most world models are built in fog. A robot learns physics from blurry cameras; a game agent infers rules it cannot read. Code is the rare world where the rules are written down.

World model. In the JEPA sense [LeCun 2022, cited above], a world model is a predictor $\hat{z}_{t+1} = g_\phi(z_t, a_t)$ of the next latent representation $z_{t+1} = \mathrm{enc}(s_{t+1})$ from the current one and an action a_t, with learned parameters ϕ. For code the world is the Form's state space and the action is an event, so the target is computed exactly by δ:

$$\mathcal{L}(\phi) = \mathbb{E}\,\big\| g_\phi\big(\mathrm{enc}(s_t), e_t\big) - \mathrm{enc}\big(\delta(s_t, e_t)\big) \big\|^2.$$

There is no aleatoric (irreducible) noise in the target. MuZero starts from the observation that lookahead planning algorithms "all rely on knowledge of the environment's dynamics, such as the rules of the game or an accurate simulator", and learns a model instead [T1, Sec. 1]; for code, those rules are δ, available in full.

Contrast with models that learn the physics. CWM, the Code World Model [B23], is a decoder mid-trained on Python execution traces and on agent trajectories. Trace prediction in that report is a learned interpreter, and the authors call it a "first step", with a neural debugger left as future work. V-JEPA 2 [B1] learns its physics from video, including 62 hours of robot video in the reported summary, and the action-conditioned predictor uses fewer than 62 hours of unlabeled robot video. Its model predictive control took about 16 seconds per action. In the Form, δ is the physics, and it is exact, so there is no aleatoric noise to fit. What a proposer still has to learn is set out in Section 7 below: intent to Form, search guidance, and the residual the ladder does not close.

Intent decoding as an inverse problem. Given an intent I stated as constraints, find a Form whose trajectories satisfy it. This is written as energy minimization in the sense of energy-based models [T36, T2]:

$$\mathcal{E}(I, f) = \alpha\, V(f) + \beta\, d\big(\tau(f), \sigma(I)\big) + \gamma\, C(f), \qquad z^\star = \arg\min_z\ \mathcal{E}\big(I, \mathrm{dec}(z)\big).$$

Here $V(f)$ is the number of checker violations (dimensionless); $\tau(f)$ the set of trajectories of f on the intent's example events; $\sigma(I)$ the trajectories the intent requires; d a distance between them (dimensionless count of mismatched

states); $C(f)$ the predicted energy per run in joules; $\alpha, \beta \geq 0$ dimensionless weights and $\gamma \geq 0$ in J^{-1}; dec the decoder from latent point to Form. The minimizer is decoded, then checked; the checker, not $\mathcal{E}$, decides acceptance. Du and Mordatch observe that "summation of two or more energies corresponds to satisfying all their goals or constraints" [T36, Sec. 1], which is why the three terms can be added. LeCun et al. add a caution: "combining two, separately trained energy-based models is not straightforward: there is no a priori guarantee that their energy scales are commensurate" [T2, Sec. 1.3, p. 6]. The weights α, β, γ must therefore be calibrated, not assumed, and their values are reported with each experiment.

Part 5. Consequences as testable predictions

A thesis that cannot lose is not a thesis. Each part above maps to a test that could prove it wrong.

- Part 1 and T2 (compression into Form) are tested by H1, H3 and H10.
- Part 3 (kinematics, energy accounting) is tested by H2, H4, H8 and H9.
- Part 4 (world model, inverse problem) is tested by H5 to H7.
- The proposer is tested by H11 to H13, on the residual of the deterministic ladder that H14 measures. H15 compares proposer classes on that residual and is unfinished. All fifteen hypotheses, with metrics and thresholds, are in one table in numerical order in Section 8 below.

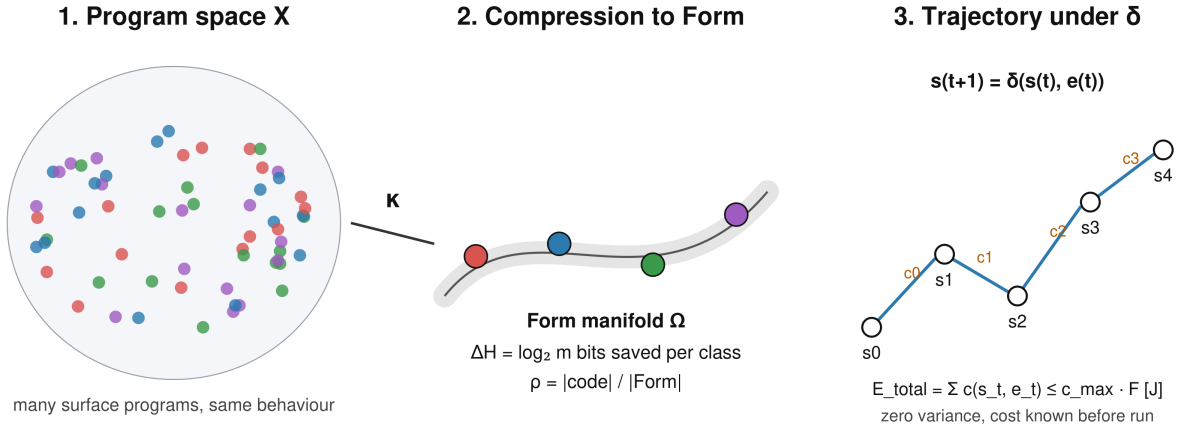

Figure 5. Information and kinematics. Left: program space, where many surface programs X express the same behaviour. Middle: canonicalization κ collapses each class onto one Form on a low-dimensional manifold Ω, saving $\Delta H = log2\ m$ bits for a class of m equally likely variants. Right: a Form moves deterministically, s(t+1) = δ(s(t), e(t)), each transition carrying a cost c in joules, with the total bounded by c_max · F before execution. A conceptual diagram, not data. .

The provable chain: hypothesis, layer, evidence

A claim about "Laws of Coding" is only as strong as its weakest link. The table names, for each hypothesis, which of the four layers of law it tests, then follows it through three layers of implementation: the algorithm (the mathematics of the Form and the predictor), the software environment (checker, corpus, harness) and the hardware (the machine and its meter). Each link carries one of four labels. **[C] cited literature:** a published result, read as stated in the ledger. **[M] measured data:** a number measured by the OpenIE authors in a run registered before it ran, with its source in a public repository. **[X] measured, not supported:** a registered measurement whose result does not support the hypothesis as intended. **[P] planned experiment:** a test not yet run. A link that is partly measured carries both [M] and [P]. Four hypotheses have measured links, as the Status section at the end of the page reports. H15 has no measured link. The four are: H1 on the hand-written programs of the Form corpus (the data section of the proposer above), H7 on its fuzzed arm only, H8 on six targets, and H14, whose three registered runs [B30, B31, B32] support it by its registered rule (Section 8 below). A measured link records that a registered run exists and what it found; it is not by itself a verdict of support. The chain is a plan for turning every remaining [P] into [M] or [X], and any [P] can still break it.

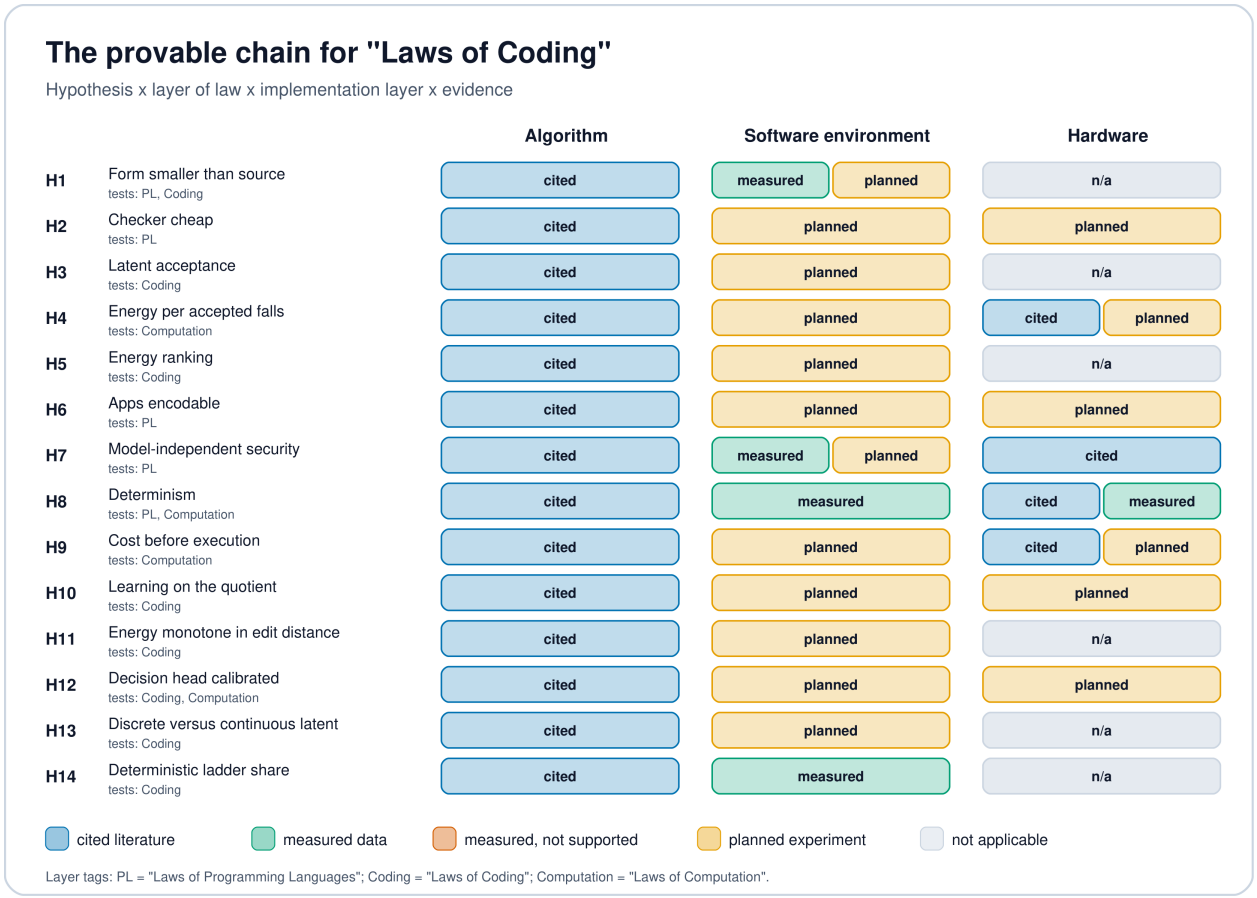

Figure 6. The provable chain for "Laws of Coding". Rows: hypotheses H1 to H14, each tagged with the layer of law it tests. H15 is not drawn on the figure. The table adds it as planned work, with no measured link. Columns: implementation layers, algorithm, software environment, hardware. Blue: cited literature. Green: measured data. Vermilion: measured, not supported. Orange: planned experiment. Grey: not applicable. The table below gives the content and source of every cell. .

ID	Hypothesis	Layer of law tested	Algorithm	Software environment	Hardware
H1	Form much smaller than source	"Laws of Programming Languages" (canonical meaning) and "Laws of Coding"	[C] Chain rule: predicting the Form saves $H(X \mid \kappa(X))$ bits [T10]	[M] Median ρ of 9.34 over 185 hand-written programs of the Form corpus against the emitted Python, above the threshold of 5, registered before the run (data section of the proposer above). [P] ρ on the canonical serialization over the lift corpus, empty until the lift pipeline is public	Not applicable (a count of bits)
H2	Checker cheap relative to generation	"Laws of Programming Languages"	[C] Validating a result can be easier than proving the generator: CompCert's register allocator (Leroy, Section 4 below)	[P] Checker wall time and cycles per candidate	[P] $C_{\text{check}}/E_{\text{model}}$ in joules on a named meter
H3	Latent prediction reaches useful acceptance	"Laws of Coding"	[C] Prediction in representation space (I-JEPA and V-JEPA 2, Section 4 below; LLM-JEPA, Huang, LeCun and Balestriero, Section 5 below)	[P] pass@1 on held-out intent-to-Form tasks	Not applicable
H4	Energy per accepted program falls	"Laws of Computation"	[C] Landauer floor $k_B T \ln 2$ per erased bit [T11, T12]	[P] J_{acc} for both arms in one harness	[C] About 3 to 4 J per token for a 65B model (Samsi et al.); memory access dominates arithmetic [T26]. [P] Device meter, idle subtracted
H5	Energy ranking favours valid Forms	"Laws of Coding"	[C] Energies add to combine constraints [T36]	[P] AUROC of energy for accepted against requires change	Not applicable

ID	Hypothesis	Layer of law tested	Algorithm	Software environment	Hardware
H6	Reference apps are encodable	"Laws of Programming Languages"	[C] No general procedure decides non-trivial semantic properties [T9, T3], hence the restricted language	[P] 7 of 7 reference apps as checked Forms	[P] Replay of the same Forms across machines (with H8)
H7	Security independent of the model	"Laws of Programming Languages"	[C] After about six CPU-years, Csmith found no wrong-code errors in CompCert (Yang et al., Section 4 below)	[M] Fuzzed arm: 0 violations of five stated invariants over 1,275 checker-accepted mutants in two runs, each registered before it ran (`rce/bench/adversarial/`). [P] Model-generated adversarial Forms, an arm that does not exist yet	[C] Speculative-execution side channels lie outside an instruction set's semantics [T26]; out of the Form's scope, stated
H8	Determinism	"Laws of Programming Languages" and "Laws of Computation"	[C] Any deterministic computation can be made logically reversible given a copy of its input [T12]	[M] Bit-identical replay of 10,500 seeded event streams with 0 mismatching lines against the reference, registered before the run (`rce/bench/determinism/`)	[C] Interrupts arrive "at unpredictable and irreproducible moments" [T8]. [M] The same run on six targets: four instruction sets (x86_64, aarch64, s390x big-endian, wasm32) and three operating systems (Linux, macOS, Windows)
H9	Cost predicted before execution	"Laws of Computation"	[C] Energy bounds speed [T14]; cost bound $c_{\max} F$ (Part 3 above)	[P] Cost-table prediction per Form	[C] Implementations of one computation differ in thermodynamic cost [T15]. [P] Median error within 20% of measured joules
H10	Learning on the quotient saves data	"Laws of Coding"	[C] Better prediction is better compression [T32]; minimum description length [T33]	[P] Equal acceptance with at most half the training examples	[P] Training energy measured for both predictors
H11	Energy monotone in edit distance	"Laws of Coding"	[C] A monotone cost ranking makes plans that deviate further from the goal cost more (Chahe et al., [B19] in the proposer section)	[P] Share of families where energy rises with edit distance	Not applicable
H12	Decision head calibrated on gate outcomes	"Laws of Coding" and "Laws of Computation"	[C] Calibrated typed decisions are measurable: pooled Choice ECE 0.028 for one hosted decision model ([B7])	[P] ECE per decision type after 10^4 receipts, against a frequency table	[P] Latency and joules per decision on a named device
H13	Discrete against continuous latent	"Laws of Coding"	[C] Discrete tokens kept long-horizon prediction stable where continuous latents accumulated error ([B3])	[P] Planning success at edit lengths 1, 5 and 20	Not applicable
H14	Deterministic ladder share	"Laws of Coding"	[C] Exact gears (lookup, rules, synthesis) run before any model in the RCE paper's cascade, with falsifier RCE-F1 on the share they close with no model call	[M] Share of a registered task set closed by the public ladder with zero model calls: three registered runs [B30, B31, B32]; run 3 closes 111 of 151 (73.5%), at or above the registered 10%, so H14 is supported by its registered rule, and 1 of 41 changes from real history (2.4%) (Section 8 below)	Not applicable
H15	Proposer classes on the residual. Unfinished	"Laws of Coding"	[P] Four classes on the 40 residual tasks, equal energy, gate acceptance. No run	[P] The comparison has not been run	[P] Equal energy when a named meter exists. No joule figure is stated

Who holds these views and why

Ideas rarely arrive alone. Each part of the thesis echoes arguments made, for their own reasons, by people who have never heard of RCE. They are listed as convergence observations: agreement in direction, never confirmation of the RCE project's claims. Where the OpenIE authors read a source in full it is quoted with its section; where they did not, the entry says so and quotes nothing.

Those who already think this way. The idea that programming and computation should have laws of their own did not originate with the RCE project, and it is older than most of the field. This page joins a conversation that has run for at least forty years and lists it as convergence, not priority.

- **Hoare and colleagues, "Laws of Programming" (1987).** Nine authors argued that "conventional procedural programs are mathematical expressions, and that they are subject to a set of laws as rich and elegant as those of any other branch of mathematics, engineering, or natural science" [T7, p. 672], and gave nearly one hundred of them. They also named the gap that remains (quoted in the timeline above) [T7, p. 686].

- **Bennett and Landauer, "The Fundamental Physical Limits of Computation" (1985).** "We are looking for general laws that must govern all information processing, no matter how it is accomplished" [T13, p. 48]. They add a caution this page shares: "To the extent that we have found limits, they are terribly far away from the real limits of modern technology" [T13, p. 48]. Their result for reversible machines: "There is thus no minimum amount of energy that must be expended in order to perform any given computation" [T13, p. 51], while erasure and error correction still dissipate, since, in their words, "Any correctional device must therefore dissipate some energy" [T13, p. 51].
- **McKee, "A few (proposed) fundamental laws of computation" (1995).** A short essay in IEEE Computer proposing laws for the field [T31]. The OpenIE authors have not obtained the text, so it is not quoted or summarized beyond its title.
- **Carette, James and Sabry, "Embracing the laws of physics" (2022).** They note that quantum theory and thermodynamics "imply that information is a conserved quantity of physical processes and hence of primitive computational operations", and set out "to redevelop foundational computational models in a way that embraces the principle of conservation of information" [T16, abstract, PDF p. 2]. Their closing summary: "The entire edifice of computer science including its mainstream models of computations, programming languages, and logics is founded on classical physics" [T16, Sec. 5.5, p. 45].
- **Lehman (1980)** states five empirical regularities of software evolution for programs embedded in the world [T6]. The first, "Continuing Change": a program "that is used and that as an implementation of its specification reflects some other reality, undergoes continual change or becomes progressively less useful" [T6, Table I, p. 1068]. He is explicit that these are not physics: "They are, however, clearly not immutable" in the way physical and chemical regularities are, since they "arise from the habits and practices of people and organizations" [T6, Sec. V-A, p. 1069], the same distance this page keeps between "Laws of Coding" and "Laws of Computation". **Brooks** distinguished "essential tasks, the fashioning of the complex conceptual structures that compose the abstract software entity" from accidental ones [T23, PDF p. 1]. **Dijkstra** observed that a computer spans a ratio of time grains of "10^{10} or more", which makes it "the first to provide us with an environment where highly hierarchical artefacts are both possible and necessary" [T8].

The groups below each hold one strand of the same idea.

Compression as understanding (Part 1, T2). Solomonoff, Kolmogorov and Chaitin defined complexity as shortest description [T27, T28, T29]; Rissanen made it a model selection rule [T33]; Hutter's prize and AIXI and Sutskever's talks extend it to intelligence (not read here; not quoted). Delétang and colleagues at DeepMind argue that "a model that compresses well generalizes well" [T32, Sec. 1] and show large language models compress images and audio. Schmidhuber argues that if the history of the universe were computable, its simplest explanation "would be the shortest program that computes it" [T34, Sec. 1]. Disagreement: Delétang et al. warn that "scaling beyond a certain point will deteriorate the compression performance since the model parameters need to be accounted for" [T32, Sec. 1], and Chollet argues that "solely measuring skill at any given task falls short of measuring intelligence" [T37, abstract]; compression of a fixed corpus is not the same as efficient learning of new skills.

Computation as physics (Part 3). Landauer [T11], Bennett [T12, T30] and Lloyd [T14] hold that information processing is a physical process with energy costs and limits. Lloyd: "Energy limits speed. Entropy limits memory." [T14, Sec. 1.5]. Wolpert's stochastic thermodynamics treats computation, including finite automata, as nonequilibrium physics: "One of the most prominent aspects of computers is that they are inherently nonequilibrium systems" [T15, abstract]; finite automata are treated in Sec. XII [T15]. Fredkin and Toffoli's conservative logic [T38], Feynman's lectures on computation, Wheeler's "it from bit" and Wolfram's computational universe point the same way (not read here). Disagreement: Bennett records objections that Landauer's principle is false, or, being a consequence of the second law of thermodynamics, either unnecessary or insufficient to exorcise Maxwell's demon, and answers them [T12]; the debate is about foundations, not about the size of the floor. A second disagreement is historical: Landauer wrote "We believe that devices exhibiting logical irreversibility are essential to computing" [T11, Sec. 3, p. 186], and Bennett's 1973 construction answered that they are not, at the price of about twice as many steps and a large temporary history [T30, pp. 525, 529]; Bennett then estimated "kT ln 2 is, as Landauer has argued [1], an approximate lower bound" only for machines that throw away about one bit per step [T30, p. 532].

State machines and deterministic semantics (Part 1, T3). Turing's machines [T9], Milner's calculus [T18], Harel's statecharts [T19], Lamport's TLA, which treats a program as a set of behaviours over states [T20], the synchronous languages Esterel and Lustre (Berry, Halbwachs) [T21, T22], Elliott's functional reactive programming [T39], the Elm architecture, deterministic simulation testing at FoundationDB and TigerBeetle, and Hoare's and Dijkstra's view of programs as predicates [T17] all treat a program as a state machine with exact semantics. They

are cited here as history. The FoundationDB and TigerBeetle pages and the Elm guide were read in full for the convergence section below (its FoundationDB, TigerBeetle and Elm Architecture rows); the others were not read in full for this section, so none of them are quoted. Limits: Turing [T9] and Rice [T3] bound what can be decided about arbitrary programs; deterministic semantics is easy to state for a closed language and hard to keep across real hardware, threads and networks.

Programs as data on a low-dimensional manifold (Part 2). Hindle et al. found software "natural", more repetitive than English [T24] (not read in full here). Alon et al.'s code2vec maps code to vectors where "semantically similar objects are mapped to close vectors" [T35, Sec. 1]. Sketch-based synthesis (Solar-Lezama), DreamCoder's library learning (Ellis et al. 2021, cited in Section 2 below; its supplement describes library learning as "automatic refactoring" [Ellis et al. 2021, supplement S3.2, p. 39]) and Allamanis's survey of machine learning on code point the same way (Solar-Lezama and Allamanis not read here). Disagreement: code2vec's authors report that "When given uninformative, obfuscated or adversarial variable names, the prediction of the label is usually less accurate" [T35, Sec. 7]: learned embeddings can track names rather than meaning, which is the gap a canonical Form is meant to close.

World models and latent prediction (Part 4). LeCun's JEPA programme at Meta FAIR (cited above), Ha and Schmidhuber, who "learn a compressed spatial and temporal representation of the environment" [T40, abstract], MuZero [T1], physics-informed neural networks, where prior physical knowledge "can act as a regularization agent that constrains the space of admissible solutions to a manageable size" [T41, Sec. 1], and Hamiltonian neural networks, motivated by models whose simulation "drifts over time to higher or lower energy states" [T42, Sec. 1], all build structure into learned dynamics. Lagrangian networks (Cranmer), Dreamer (Hafner) and Koopman operator learning share the aim (not read here). Disagreement: Ha and Schmidhuber found that a learned world model "will be exploitable by the controller, even if in the actual environment such exploits do not exist" [T40, Sec. 4.5]; Greydanus et al. note their method assumes a conserved quantity and so cannot model friction [T42, Sec. 3.2]. For code, the checker is the guard against exploiting the model's errors.

Energy-based models (Part 4). Hopfield [T43] and Hinton [T44] framed learning as shaping an energy surface (not read here); Du and Mordatch [T36] show implicit generation by descending a learned energy, and LeCun (cited above) places energy minimization at the centre of JEPA inference. Disagreement within the field concerns training stability and sampling cost, which Du and Mordatch discuss [T36].

Lane 3, seen from outside the project

Section 3 below records programmes that reconceive what computation, programming or coding is. They are convergence observations. The commitment section below maps some of the same names onto the seven commitments, one by one.

- **Unison.** A definition is the hash of its syntax tree. Names are a view. Two texts with the same tree are the same definition. That is the identity half of a canonical Form.
- **egg.** An e-graph holds a set of equivalent programs as one structure (Willsey et al., Section 3 below). The Laws of Coding use the same quotient: many surface programs, one Form.
- **Interaction nets.** HVM2 (the Higher Order Company's interaction-combinator model) treats computation as local rewriting, with parallelism inherent in the model. Section 3 cites the model and does not carry that paper's figures.
- **Proof assistants.** Lean, Dafny and Verus make correctness a checkable object. The vericoding benchmark is the measured floor cited in Section 3. AlphaProof, DeepSeek-Prover-V2, Kimina-Prover Preview and Goedel-Prover use the Lean kernel as the gate. In the reports read here, the domain is mathematics.
- **Elliott, denotational design.** "The instance's meaning is the meaning's instance." A semantic model defines correctness and carries it to the implementation (Elliott, 2009, revised 2016). It is a manual discipline. It is the same demand that a Form have one meaning.
- **Propagators.** Radul and Sussman (2009) describe computation as autonomous machines adding information to shared cells, unifying expressions and constraints. The report is theory with a Scheme implementation.
- **Lean, as an engineering programme.** The Lean FRO (Focused Research Organization) Year 1 roadmap, August 2023 to July 2024, lists a standard library, a module system, Lake and Reservoir, proof automation, machine-learning integration and a new code generator. Later years are not used here.
- **Chollet.** Intelligence as skill-acquisition efficiency, with the Abstraction and Reasoning Corpus (ARC) as the measure [T37]. The ARC Prize results in Section 3 are the later measurement of that programme. Talks

after 2019 are not cited on their own.

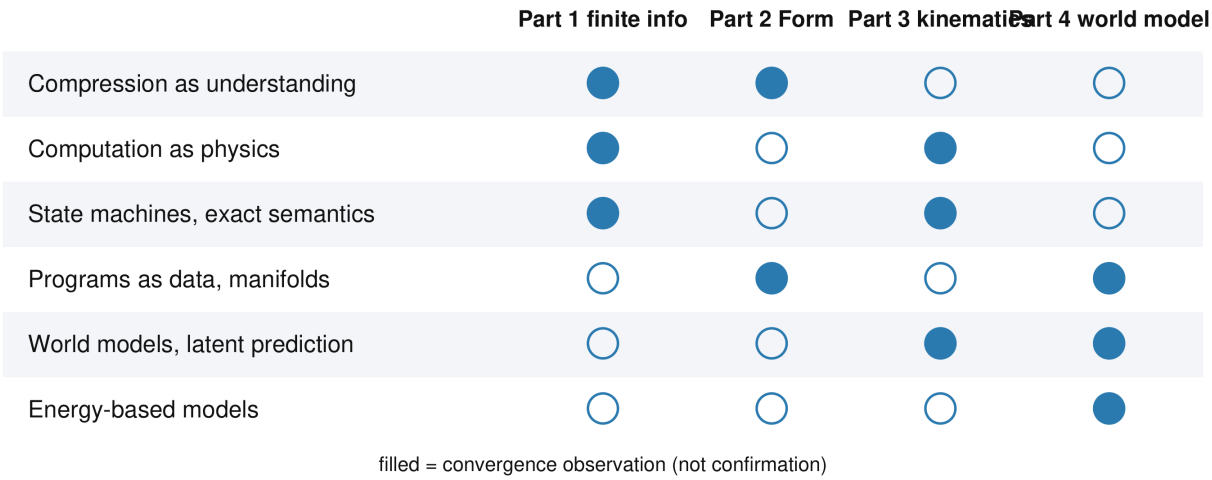

Figure 7. Which communities bear on which part of the thesis. A filled dot marks a convergence observation, not confirmation. Part 1: finite information system; T2: compression into Form; T3: kinematics and energy; T4: world model and inverse problem. .

Thesis references

Ledger: Full = read in full; Partial = read in part; Unavailable = not read (paywalled or not retrieved), listed with identifier only and never quoted.

1. [T1] J. Schrittwieser et al., "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model", Nature 588 (2020). arXiv:1911.08265. Full.
2. [T2] Y. LeCun et al., "A tutorial on energy-based learning", 2006. PDF. Full.
3. [T3] H. G. Rice, "Classes of recursively enumerable sets and their decision problems", Trans. AMS 74 (1953). doi:10.1090/S0002-9947-1953-0053041-6. Unavailable.
4. [T4] M. A. Di Liscia, "Johannes Kepler", Stanford Encyclopedia of Philosophy. plato.stanford.edu. Full.
5. [T5] G. E. Smith, "Newton's Philosophiae Naturalis Principia Mathematica", Stanford Encyclopedia of Philosophy. plato.stanford.edu. Full.
6. [T6] M. M. Lehman, "Programs, life cycles, and laws of software evolution", Proceedings of the IEEE 68(9) (1980). doi:10.1109/PROC.1980.11805. Full.
7. [T7] C. A. R. Hoare, I. J. Hayes, He Jifeng, C. C. Morgan, A. W. Roscoe, J. W. Sanders, I. H. Sorensen, J. M. Spivey and B. A. Sufrin, "Laws of Programming", Communications of the ACM 30(8), 672-686 (1987). doi:10.1145/27651.27653; PDF copy. Full.
8. [T8] E. W. Dijkstra, "The Humble Programmer", ACM Turing Lecture 1972, EWD 340. Transcription; doi:10.1145/355604.361591. Full.
9. [T9] A. M. Turing, "On computable numbers", Proc. London Math. Soc. 42 (1936). doi:10.1112/plms/s2-42.1.230. Unavailable.
10. [T10] C. E. Shannon, "A mathematical theory of communication", Bell System Technical Journal 27 (1948). doi:10.1002/j.1538-7305.1948.tb01338.x; reprint (page numbers refer to this reprint). Full (all 55 pp. of the reprint, Parts I to V and Appendices 1 to 7).
11. [T11] R. Landauer, "Irreversibility and heat generation in the computing process", IBM J. Res. Dev. 5 (1961). doi:10.1147/rd.53.0183. Full.
12. [T12] C. H. Bennett, "Notes on Landauer's principle, reversible computation, and Maxwell's Demon", 2003. arXiv:physics/0210005. Full.
13. [T13] C. H. Bennett and R. Landauer, "The Fundamental Physical Limits of Computation", Scientific American 253(1), 48-57 (1985). doi:10.1038/scientificamerican0785-48; PDF copy. Full (all text pages of the linked copy, pp. 48-56).
14. [T14] S. Lloyd, "Ultimate physical limits to computation", Nature 406 (2000). arXiv:quant-ph/9908043. Full.

15. [T15] D. H. Wolpert, "The stochastic thermodynamics of computation", J. Phys. A 52 (2019). arXiv:1905.05669. Full.
16. [T16] J. Carette, R. P. James and A. Sabry, "Embracing the laws of physics: Three reversible models of computation", Advances in Computers (2022). doi:10.1016/bs.adcom.2021.11.009; author copy. Full.
17. [T17] C. A. R. Hoare, "An axiomatic basis for computer programming", CACM 12 (1969). doi:10.1145/363235.363259. Unavailable.
18. [T18] R. Milner, A Calculus of Communicating Systems, LNCS 92 (1980). doi:10.1007/3-540-10235-3. Unavailable.
19. [T19] D. Harel, "Statecharts: a visual formalism for complex systems", Sci. Comput. Program. 8 (1987). doi:10.1016/0167-6423(87)90035-9. Unavailable.
20. [T20] L. Lamport, "The temporal logic of actions", ACM TOPLAS 16 (1994). doi:10.1145/177492.177726. Unavailable.
21. [T21] G. Berry and G. Gonthier, "The Esterel synchronous programming language", Sci. Comput. Program. 19 (1992). doi:10.1016/0167-6423(92)90005-V. Unavailable.
22. [T22] N. Halbwachs et al., "The synchronous data flow programming language LUSTRE", Proc. IEEE 79 (1991). doi:10.1109/5.97300. Unavailable.
23. [T23] F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer 20(4) (1987; first presented 1986). doi:10.1109/MC.1987.1663532; PDF copy. Full.
24. [T24] A. Hindle et al., "On the naturalness of software", ICSE (International Conference on Software Engineering) 2012. doi:10.1109/ICSE.2012.6227135. Unavailable.
25. [T25] J. W. Backus et al., "The FORTRAN Automatic Coding System", Western Joint Computer Conference (1957). Computer History Museum copy. Full.
26. [T26] J. L. Hennessy and D. A. Patterson, "A New Golden Age for Computer Architecture", Communications of the ACM 62(2) (2019). doi:10.1145/3282307; CACM web text. Full.
27. [T27] R. J. Solomonoff, "A formal theory of inductive inference, Part I", Information and Control 7 (1964). doi:10.1016/S0019-9958(64)90223-2. Unavailable.
28. [T28] A. N. Kolmogorov, "Three approaches to the quantitative definition of information", Problems of Information Transmission 1 (1965); English reprint, International Journal of Computer Mathematics 2 (1968), doi:10.1080/00207166808803030. Full (page numbers from the 1968 reprint).
29. [T29] G. J. Chaitin, "On the length of programs for computing finite binary sequences", J. ACM 13 (1966). doi:10.1145/321356.321363. Unavailable.
30. [T30] C. H. Bennett, "Logical Reversibility of Computation", IBM J. Res. Dev. 17(6), 525-532 (1973). doi:10.1147/rd.176.0525. Full.
31. [T31] G. McKee, "A Few (Proposed) Fundamental Laws of Computation", IEEE Computer 28(1), 120 (1995), Open Channel. No DOI located; DBLP table of contents. Unavailable.
32. [T32] G. Delétang et al., "Language Modeling Is Compression", ICLR 2024. arXiv:2309.10668. Full.
33. [T33] J. Rissanen, "Modeling by shortest data description", Automatica 14 (1978). doi:10.1016/0005-1098(78)90005-5. Unavailable.
34. [T34] J. Schmidhuber, "Driven by Compression Progress", 2009. arXiv:0812.4360. Full.
35. [T35] U. Alon et al., "code2vec: Learning Distributed Representations of Code", POPL 2019. arXiv:1803.09473. Full.
36. [T36] Y. Du and I. Mordatch, "Implicit Generation and Modeling with Energy-Based Models", NeurIPS 2019. arXiv:1903.08689. Full.
37. [T37] F. Chollet, "On the Measure of Intelligence", 2019. arXiv:1911.01547. Full.
38. [T38] E. Fredkin and T. Toffoli, "Conservative logic", Int. J. Theor. Phys. 21 (1982). doi:10.1007/BF01857727. Unavailable.
39. [T39] C. Elliott and P. Hudak, "Functional reactive animation", ICFP 1997. doi:10.1145/258948.258973. Unavailable.
40. [T40] D. Ha and J. Schmidhuber, "World Models", 2018. arXiv:1803.10122. Full.
41. [T41] M. Raissi, P. Perdikaris and G. E. Karniadakis, "Physics Informed Deep Learning (Part I)", 2017. arXiv:1711.10561. Full.
42. [T42] S. Greydanus, M. Dzamba and J. Yosinski, "Hamiltonian Neural Networks", NeurIPS 2019. arXiv:1906.01563. Full.

43. [T43] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities”, PNAS 79 (1982). doi:10.1073/pnas.79.8.2554. Unavailable.
44. [T44] G. E. Hinton, “Training products of experts by minimizing contrastive divergence”, Neural Computation 14 (2002). doi:10.1162/089976602760128018. Unavailable.

Convergence by commitment

The RCE thesis rests on seven commitments. Each has been reached independently, often more than once, by people working for their own reasons. The tables below list projects whose primary sources (paper, official documentation or repository) were opened and checked against the sentence written here. Each row says what the project established and where it stops relative to the commitment. These are convergence observations: agreement in direction, not confirmation of RCE’s claims and not a ranking. Items from OpenIE, the organization behind RCE, are marked *same house* and are never used as evidence.

Acronyms used in this section: RCE, the Reality Code Engine; AST, abstract syntax tree; IR, intermediate representation; LLM, large language model; JSON, JavaScript Object Notation; WASM, WebAssembly; SMT, satisfiability modulo theories (Z3 is an SMT solver); DST, deterministic simulation testing; CEGIS, counterexample-guided inductive synthesis.

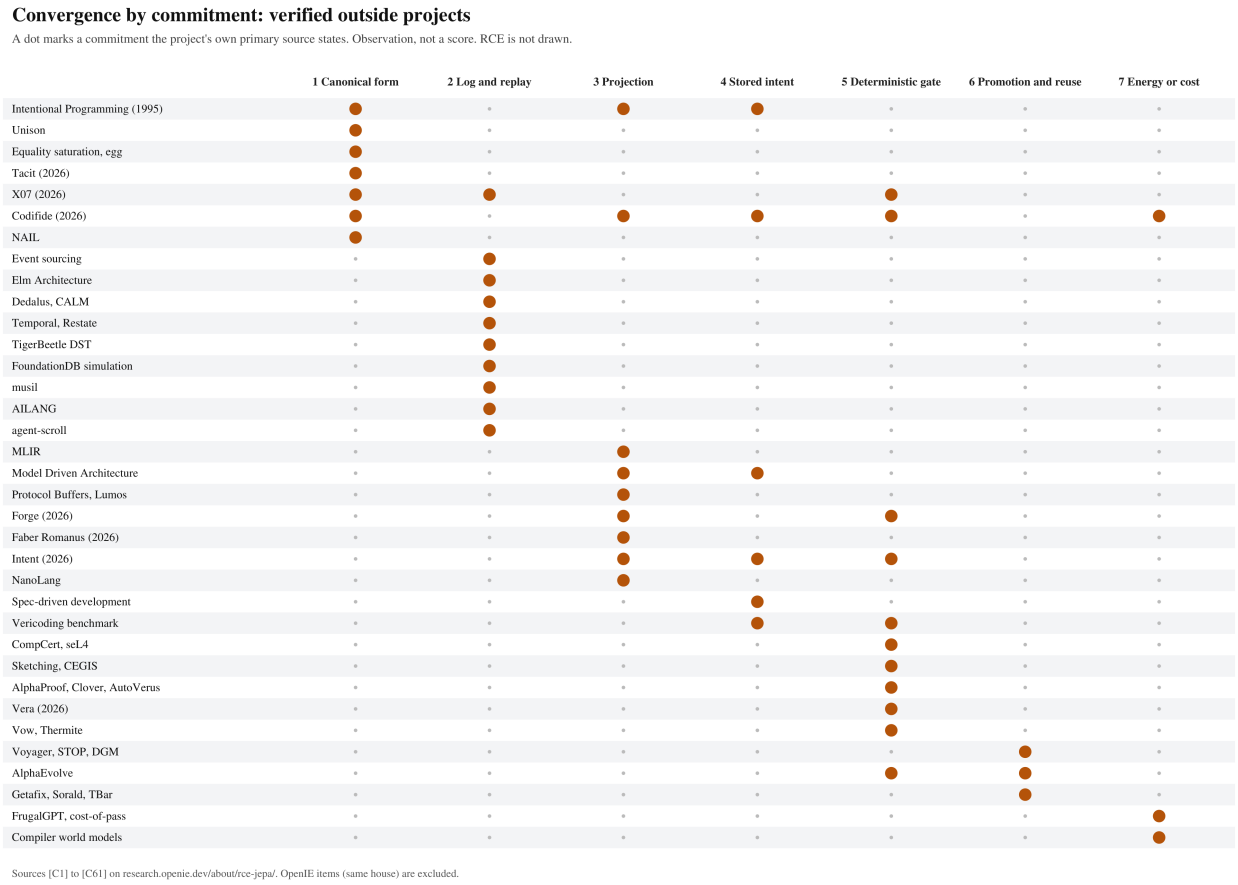

Figure 8. Which commitment each verified outside project reaches, according to its own primary source. A dot is a convergence observation, not a score; RCE is not drawn. Select the figure to enlarge it. .

1. One canonical form; surface text is a view

Project	What it established	Where it stops
Intentional Programming, Simonyi, 1995 [C1]	A program is a tree of nodes, each an instance of an intention; users edit the tree and view it in any notation; computational intent can be kept separate from implementation	A research system at Microsoft; the report describes no determinism or replay model (a transaction file feeds version control, but no replay semantics are given)
Unison [C2]	Each definition is identified by a hash of its syntax tree; names are separately stored metadata	The hash is of structure: two definitions with the same behaviour and different structure have different hashes
Equality saturation and egg [C3; C4]	An e-graph holds many equivalent versions of a program at once; a cost model picks one (Tate et al.: a pseudo-Boolean solver over a static cost model; egg: an extractor with a user-provided cost function)	An optimization and synthesis technique applied per function (intraprocedural in Tate et al.) or over terms in a user-defined language (egg, which has no built-in support for binding); saturation is bounded when it does not terminate, and the e-graph is not kept as the program's stored form
Tacit, 2026 [C5]	The AST is the source of truth; every valid AST has one byte-exact canonical text; definitions are addressed by BLAKE3 hash; malformed code becomes typed Hole nodes	One target (LLVM); no state-machine semantics stated
X07, 2026 [C6]	Canonical source is a JSON AST, edited with JSON Patch; a 646-code diagnostic catalogue with enforced quickfix coverage	Canonical bytes, not canonical behaviour
Codifide, 2026 [C7]	The canonical form is a typed hypergraph with a deterministic byte form and SHA-256 hash; surface syntax is a projection; intent is part of a definition's identity	Described by its authors as a contract-and-dispatch language, not general computation
NAIL [C8]	`nail canonicalize` normalizes any program; `nail check --strict` rejects non-canonical input	An RFC 8785-inspired JSON subset that, in its own words, does not claim full RFC 8785 compliance; byte canonicalization only
EqNet, Allamanis et al., 2017 [C9]	Neural representations trained on known equivalence classes so that equivalent expressions get identical or highly similar vectors	Boolean and polynomial expressions (supervised), not programs

2. A program is a deterministic state machine over a log

Project	What it established	Where it stops
Abstract State Machines, Gurevich, 2000 [C10]	The paper's title states the result: sequential abstract state machines capture sequential algorithms	A theorem, not a system: every sequential algorithm, defined by three postulates (sequential time, abstract state, bounded exploration; Def. 6.1), has an equivalent sequential abstract state machine that matches it step for step (Main Theorem 6.13). That the postulates cover all sequential algorithms is argued, not proved
Event sourcing, Fowler, 2005 [C11]	"Capture all changes to an application state as a sequence of events"	A pattern, published as a draft; replay stays consistent only if gateways record external query responses and suppress external updates during replay
The Elm Architecture [C12]	Model, View, Update: state, a view of the state, and an update from messages	One language and one target (the browser)
Dedalus and the CALM theorem, Hellerstein, Alvaro et al. [C13; C14]	Dedalus: Datalog with explicit time, modelling mutable state and asynchronous messaging. CALM: a program has a consistent, coordination-free distributed implementation if and only if it is monotonic (proved by Ameloot et al.; [C14] gives a proof sketch)	Dedalus and Bloom are research languages and CALM is a theorem; neither emits to other languages
Durable execution: Temporal, Restate [C15; C16]	Temporal: workflow code "must be deterministic to support replay". Restate: every step and its result are recorded in a journal	Replay serves resilience, not program identity
TigerBeetle deterministic simulation testing, 2026 [C17]	Logical determinism (like FoundationDB) plus physical determinism: replicas converge to byte-by-byte identical state; real code runs in a simulator	One database, built by hand to this discipline
FoundationDB simulation testing [C18]	"A deterministic simulation of an entire FoundationDB cluster within a single-threaded process"; determinism "allows perfect repeatability of a simulated run"	A testing discipline for one database, not a program form

Project	What it established	Where it stops
musil [C19]	An explicit-state model checker as a Python library; its simulate mode runs the real node code on a seeded, replayable simulated network	Bounded state spaces; it checks the model, and simulation finds bugs rather than proving their absence
AILANG; agent-scroll [C20; C21]	AILANG: every construct has deterministic semantics, with replayable execution and structured traces. agent-scroll: canonical, byte-deterministic, hash-chained transcripts with a verifier that catches any single-byte change	AILANG is functional, not a state-machine form; agent-scroll logs conversations, not programs

3. Every target is a projection: write once, emit many

Project	What it established	Where it stops
MLIR [C22]	A reusable compiler infrastructure spanning levels of abstraction, hardware targets and domains	Starts below the source: it has no parser generation or AST construction (Sec. 7), and its paper names translation validation across an extensible dialect ecosystem as an open problem (Sec. 2)
Model Driven Architecture, OMG, 2003 [C23]	Computation-independent, platform-independent and platform-specific viewpoints	Petre reports, from Forward and Lethbridge, that keeping the model up to date with the code was the biggest perceived problem (68% agreement) [C24]
Protocol Buffers; Lumos [C25; C26]	Define a schema once and generate bindings in many languages (Lumos: Rust and TypeScript for Solana)	Data shapes, not behaviour; the same Protocol Buffers message can have many binary serializations ("You cannot compare two messages for equality without fully parsing them")
Forge, 2026 [C27]	"Many frontends feed one verified intermediate representation, and that representation feeds many backends"	Board and peripheral descriptions, not general programs
Faber Romanus, 2026 [C28]	One high-level IR projected to Rust, Go, TypeScript, Swift and Faber, and through a lower IR to LLVM and WASM text, with Metal and CUDA kernels; support is measured and stated per target	Agreement across targets is measured by corpus coverage, not proved
Intent, 2026 [C29]	One source file compiles to Rust, JavaScript and WASM; contracts are enforced at runtime in every target and optionally checked by Z3	Three targets
NanoLang [C30]	Core semantics proved in Coq with no axioms; C is the production target	PTX and RISC-V are experimental; LLVM and WASM are planned

4. Store intent; derive the implementation

Project	What it established	Where it stops
FORTRAN, Backus et al., 1957; deductive synthesis, Manna and Waldinger, 1980 [C31; C32]	Automatic coding from formulas; programs extracted from constructive proofs	FORTRAN became a compiler; deductive synthesis had no implementation at the time
Spec-driven development, as classified by Böckeler [C33]	Three levels: spec-first, spec-anchored and spec-as-source, in which "only the spec is edited by the human, the human never touches the code"; of Kiro, spec-kit and Tessl, only Tessl explores spec-as-source	The spec is natural language, and Böckeler saw code generated repeatedly from one spec differ; nothing checks that the code is the spec
Vericoding benchmark, 2025 [C34]	12,504 formal specifications in Dafny, Verus and Lean; at least one of nine off-the-shelf LLMs, with five attempts each, solved 82% (Dafny), 44% (Verus) and 27% (Lean) of the tasks, and the best single model reached 67.5%, 30.9% and 17.9% (Table 3); adding natural-language descriptions gave no statistically significant gain on the one subset tested (Verina, 157 tasks; Sec. 4.1)	Specification languages are the proof languages; no runtime form or multi-target emission; among successes about 9% of specs were too weak to capture intent (Sec. 4)
Codifide [C7]	Every definition carries an intent that compilation never discards	Contract-and-dispatch scope

Recommendation. The vericoding benchmark [C34] is used as an outside benchmark for the step from specification to Form: a slice of its state-machine-shaped Dafny, Verus and Lean specifications is lifted into Forms and the

share the RCE gate accepts is reported beside the published vericoding results.

5. The model proposes; a deterministic gate decides

Project	What it established	Where it stops
CompCert; seL4 [C35; C36]	Machine-checked compiler and kernel; CompCert checks an unverified heuristic's output with a verified checker; seL4's proof cost about 20 person-years	Proof effort per artifact
Sketching, Solar-Lezama et al., 2006 (the counterexample-driven loop later named CEGIS) [C37]	Against a human-written partial program (sketch) and an executable specification, a synthesizer proposes, a checker returns a counterexample, and the loop repeats	Proposers are SAT solvers, not learned models; limited to finite programs (bounded input, guaranteed termination), with guarantees only for the bounded instance
AlphaProof; Clover; AutoVerus [C38; C39; C40]	A model proposes; Lean, Dafny or Verus checks	Gate on proofs or code text; no canonical program form
Agentic model checking, 2026; LLM-driven verified synthesis for concept design, 2026 [C41; C42]	"Agents propose, solvers verify" (an LLM still triages counterexamples for realism); in the second, invariants alone under-constrained designs, which differed across runs, some of them implausible	[C41] verifies existing code with LLM-inferred specs; checks are bounded (default unwind 4) and spec-relative, and the authors name over-weak specs that "cause bugs to be missed silently" as the primary failure mode. [C42] synthesizes coordination rules for design models (not code), checked only within a bounded Alloy scope, with an LLM translating the design into Alloy. Both warn that validity is not intent
Vera, 2026; Vow; Thermite [C43; C44; C45]	Vera: mandatory contracts, proved by the Z3 SMT solver where decidable and otherwise checked at run time: "The model doesn't need to be right, it needs to be checkable". Vow: contracts statically verified by ESBMC bounded model checking. Thermite: contracts with counterexamples on failure and a recorded assurance level (Verus/Z3, Lean, Kani bounded, runtime)	Gate at the function; no replay or promotion
Decision models, a class of concept [C46]	A model whose entire output is a typed answer (one option from a closed set, a bounded score, or a yes/no probability) with a probability for every option, trained for calibration	Needs a closed option set. The RCE project builds its own decision head and uses no outside decision model

6. Accepted work is promoted and reused

Project	What it established	Where it stops
Gödel machine, Schmidhuber; Löbian obstacle, Yudkowsky and Herreshoff [C47; C48]	Self-rewrite only after a proof, under its own axioms, that the rewrite has higher utility than continuing to search; in the straightforward formalism, a proof-based agent cannot trust a successor that uses its own proof system (Löb's theorem), so trust shrinks with each rewrite, and the paper's technical workarounds are only partial	Theory
Voyager; STOP; Darwin Gödel Machine; AlphaEvolve [C49; C50; C51; C52]	A growing library of executable skills; scaffolds that improve themselves (the model's weights are unchanged); archives of self-modified agents (kept if they still compile and can edit code; parents chosen by benchmark score); code evolved under automatic evaluators	Selection by score or evaluator, not by proof at an interface. Voyager's evaluator is a GPT-4 critic; AlphaEvolve's scores may include LLM-graded feedback, and its correctness came from construction, randomized testing or human and verification review outside the loop
Getafix; Sorald; TBar [C53; C54; C55]	Fix patterns mined from past fixes or written as templates, reused on new cases without a model; with perfect fault localization TBar correctly fixes 74 of 395 Defects4J bugs (plausible patches for 101), and 43 with standard fault localization	Patterns are per bug class, learned or written for one language (Getafix's method itself is language-agnostic); Getafix checks only its top fix, by re-running the static analyzer

7. Energy or cost is the objective

Project	What it established	Where it stops
Landauer, 1961; Bennett, 1973 [C56; C57]	Erasing a bit has a thermodynamic floor; computation can in principle be made logically reversible	Physics, not a software objective
FrugalGPT, 2023 [C58]	A cascade of models matched the best single model with up to 98% lower cost on its tasks	Dollar cost; no deterministic tiers below the models
Cost-of-pass, 2025; AI Agents That Matter, 2024 [C59; C60]	Expected monetary cost of a correct solution; report and jointly optimize cost with accuracy	Dollars and tokens, not joules; the Cost-of-Pass authors note energy could replace dollars but do not measure it
Compiler world models, 2026 [C61]	Rolls a tensor program's state forward through scheduling actions in latent space, then ranks candidate schedules within a workload before measuring on hardware (a relative score, not a latency prediction)	Tensor schedules only (TVM AutoScheduler); ranks candidates, does not generate them
Codifide [C7]	Selects among satisfied candidates by declared cost	Declared cost, not measured energy
JouleOS (OpenIE) [same house]	Energy budgets and the `est_j`, `reported_j`, `measured_j` labels	Same house: listed for disclosure, not as evidence

How four commitments meet the later lanes

Section 3 below records three lanes. Figure 8 is unchanged. The notes here are convergence readings of sources already in the tables, plus the lane reports, not a score.

- **One canonical form.** Unison [C2] identifies a definition by a hash of its syntax tree, and names are a view. egg [C4] stores equivalent programs as one e-graph. Both treat structure, not surface text, as the object.
- **The model proposes and a deterministic gate decides.** AlphaEvolve [C52] and AlphaProof [C38] are hybrid instances, together with the vericoding benchmark [C34]. Section 3 adds the Lean provers DeepSeek-Prover-V2, Kimina-Prover Preview and Goedel-Prover, where the Lean kernel is the gate, and type-constrained decoding, where the type system acts while the model samples.
- **Promotion and reuse.** DreamCoder (Ellis et al., Section 2 below) learns a library by compression. LILO (Grand et al., Section 3 below) pairs a large language model with symbolic refactoring. The library is the reuse step in those systems. The domains they report are small.
- **Cost.** In the CWM report, a best of 16 samples at test time moves SWE-bench Verified pass@1 from 53.9% to 65.8% [B23]. That is a compute cost on one benchmark, not a joule measurement. Mercury reports throughput of 1,109 and 737 tokens per second on an H100 (Inception Labs, Section 3 below). Throughput is not joules. This page still counts cost as electrical energy, idle subtracted when a meter reading exists, and does not convert those figures into joules.

What the RCE paper states (OpenIE's own work)

The following are from the RCE paper (revision df38742, 6 November 2026). They are listed here as the project's own work, not as a comparison with the projects above.

- **Canonical form.** The canonical serialization is injective on Forms, so two Forms that differ have different shape hashes. Equivalence of behaviour is defined separately (fit equivalence), and in the engine's design equivalent variants are grouped by sound rewrites in an e-graph; the paper does not claim that equivalent behaviour always yields one canonical form.
- **Proofs.** Determinism, totality, replay as a fold of the log, and atomicity of a kept step (an event that does not apply keeps the state) are proved in Lean 4 on a core calculus. The Rust program emitter is proved on a core as a verified translation into MiniRust, a deep embedding of the emitted Rust subset; that the Rust compiler implements MiniRust is trusted and tested. This proof covers the program emitter, not the Form projection.
- **Gate.** In the engine's design, a candidate passes replay of recorded inputs, a shadow run with a coverage bound, an output schema check and a Kani proof certificate, and a proposed rewrite also needs a Z3 proof. In the public record of one run, a local 1.5-billion-parameter model made 18 proposals, 13 of them distinct; the gate did not admit 15, and the 3 it admitted are one rewrite under three names that repeats a rule the engine already had.

- **Energy.** In the engine's design every receipt carries `est_j`; an unpriced model call is never read as zero. `measured_j` (wall meter) is empty in every result so far.

Sources for this section

Labels as in the ledger: Full = the whole primary source (paper, documentation page, README or project description) was read, including references and appendices where present. Every source listed here was read in full on or before 14 October 2026; [C46] points to sources listed in the decision-head section.

1. [C1] C. Simonyi, "The Death of Computer Languages, the Birth of Intentional Programming", Microsoft Research MSR-TR-95-52, September 1995. Microsoft Research; PDF copy. Full.
2. [C2] Unison, "The big idea". unison-lang.org. Full.
3. [C3] R. Tate, M. Stepp, Z. Tatlock and S. Lerner, "Equality Saturation: A New Approach to Optimization", Logical Methods in Computer Science 7(1:10), 2011 (extended version of POPL 2009). arXiv:1012.1802. Full.
4. [C4] M. Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock and P. Panchekha, "egg: Fast and Extensible Equality Saturation", POPL 2021. arXiv:2004.03082. Full.
5. [C5] Tacit. github.com/weetster/tacit. Full (README).
6. [C6] X07. github.com/x07lang/x07. Full (README).
7. [C7] Codifide. github.com/codifide/codifide-programming-language. Full (README).
8. [C8] NAIL. pypi.org/project/nail-lang. Full (project description).
9. [C9] M. Allamanis, P. Chanthirasegaran, P. Kohli and C. Sutton, "Learning Continuous Semantic Representations of Symbolic Expressions", ICML 2017 (PMLR 70). arXiv:1611.01423. Full.
10. [C10] Y. Gurevich, "Sequential Abstract State Machines Capture Sequential Algorithms", ACM TOCL 1(1), 2000. doi:10.1145/343369.343384; PDF copy. Full.
11. [C11] M. Fowler, "Event Sourcing", 2005. martinfowler.com. Full.
12. [C12] The Elm Architecture. guide.elm-lang.org. Full.
13. [C13] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier and R. Sears, "Dedalus: Datalog in Time and Space", UC Berkeley EECS-2009-173, December 2009. PDF. Full.
14. [C14] J. M. Hellerstein and P. Alvaro, "Keeping CALM: When Distributed Consistency is Easy", 2019. arXiv:1901.01930. Full. The formal proof is T. J. Ameloot, F. Neven and J. Van den Bussche, "Relational transducers for declarative networking", JACM 60(2), 2013.
15. [C15] Temporal, "Workflow Definition". docs.temporal.io. Full.
16. [C16] Restate, "Key Concepts: Durable Execution". docs.restate.dev. Full.
17. [C17] C. Bhandari, "Protocol-Aware Deterministic Simulation Testing", TigerBeetle, 20 August 2026. tigerbeetle.com. Full.
18. [C18] FoundationDB, "Simulation and Testing". apple.github.io/foundationdb. Full.
19. [C19] musil (project description, version 0.35.6). pypi.org/project/musil. Full.
20. [C20] AILANG. github.com/sunholo-data/ailang. Full (README).
21. [C21] agent-scroll. github.com/p-vbordei/agent-scroll. Full (README).
22. [C22] C. Lattner et al., "MLIR: A Compiler Infrastructure for the End of Moore's Law", 2020. arXiv:2002.11054. Full.
23. [C23] OMG, MDA Guide Version 1.0, 2003. PDF. Full.
24. [C24] M. Petre, "UML in practice", ICSE 2013. author copy. Full.
25. [C25] Protocol Buffers overview. protobuf.dev. Full.
26. [C26] Lumos introduction. docs.lumos-lang.org. Full.
27. [C27] Forge. github.com/Sanjith-Shan/Forge. Full (README).
28. [C28] Faber Romanus. github.com/faberlang/faber. Full (README).
29. [C29] Intent. github.com/lhaig/intent. Full (README).
30. [C30] NanoLang. github.com/jordanhubbard/nanolang. Full (README).
31. [C31] J. W. Backus et al., "The FORTRAN Automatic Coding System", 1957. Computer History Museum copy. Full.
32. [C32] Z. Manna and R. Waldinger, "A Deductive Approach to Program Synthesis", ACM TOPLAS 2(1), 1980. doi:10.1145/357084.357090. Full.

33. [C33] B. Böckeler, “Understanding Spec-Driven-Development: Kiro, spec-kit, and Tessl”, martinfowler.com, 15 October 2025. martinfowler.com. Full.
34. [C34] S. Bursuc, T. Ehrenborg, S. Lin et al., “A benchmark for vericoding: formally verified program synthesis”, 2025. arXiv:2509.22908. Full.
35. [C35] X. Leroy, “Formal verification of a realistic compiler”, CACM, 2009. PDF. Full.
36. [C36] G. Klein et al., “seL4: Formal Verification of an OS Kernel”, SOSP 2009. PDF. Full.
37. [C37] A. Solar-Lezama, L. Tancau, R. Bodik, V. Saraswat and S. Seshia, “Combinatorial Sketching for Finite Programs”, ASPLOS 2006. PDF. Full.
38. [C38] T. Hubert et al., “Olympiad-level formal mathematical reasoning with reinforcement learning”, Nature, 2025. Nature. Full.
39. [C39] C. Sun et al., “Clover: Closed-Loop Verifiable Code Generation”. arXiv:2310.17807. Full.
40. [C40] C. Yang et al., “AutoVerus”. arXiv:2409.13082. Full.
41. [C41] Y. Sun, J. Liu, D. Kroening and J. Xue, “Agentic Model Checking”, 2026. arXiv:2605.21434. Full.
42. [C42] A. Cunha, “Verified LLM-Driven Synthesis for Concept Design”, 2026. arXiv:2607.15718. Full.
43. [C43] Vera. github.com/aallan/vera. Full (README).
44. [C44] Vow. github.com/vow-lang/vow. Full (README).
45. [C45] Thermite. github.com/dollspace-gay/Thermite. Full (README).
46. [C46] Decision models as a class: see the decision-head section above and its sources [B7; B8; B12].
47. [C47] J. Schmidhuber, “Goedel Machines: Self-Referential Universal Problem Solvers Making Provably Optimal Self-Improvements”, TR IDSIA-19-03 (v5, 2006). arXiv:cs/0309048. Full.
48. [C48] E. Yudkowsky and M. Herreshoff, “Tiling Agents for Self-Modifying AI, and the Löbian Obstacle”, MIRI early draft, 7 October 2013. PDF. Full.
49. [C49] G. Wang et al., “Voyager: An Open-Ended Embodied Agent with Large Language Models”, 2023. arXiv:2305.16291. Full.
50. [C50] E. Zelikman, E. Lorch, L. Mackey and A. T. Kalai, “Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation”, COLM 2024. arXiv:2310.02304. Full.
51. [C51] J. Zhang, S. Hu, C. Lu, R. Lange and J. Clune, “Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents”, ICLR 2026. arXiv:2505.22954. Full.
52. [C52] A. Novikov et al., “AlphaEvolve: A coding agent for scientific and algorithmic discovery”, Google DeepMind white paper, 2025. arXiv:2506.13131. Full.
53. [C53] J. Bader, A. Scott, M. Pradel and S. Chandra, “Getafix: Learning to Fix Bugs Automatically”, OOPSLA 2019. arXiv:1902.06111. Full.
54. [C54] K. Etemadi et al., “Sorald: Automatic Patch Suggestions for SonarQube Static Analysis Violations”, 2021 (v2 2022). arXiv:2103.12033. Full.
55. [C55] K. Liu et al., “TBar: Revisiting Template-based Automated Program Repair”, 2019. arXiv:1903.08409. Full.
56. [C56] R. Landauer, “Irreversibility and heat generation in the computing process”, 1961. doi:10.1147/rd.53.0183. Full.
57. [C57] C. H. Bennett, “Logical Reversibility of Computation”, 1973. doi:10.1147/rd.176.0525. Full.
58. [C58] L. Chen, M. Zaharia and J. Zou, “FrugalGPT”, 2023. arXiv:2305.05176. Full.
59. [C59] M. H. Erol, B. El, M. Suzgun, M. Yüksekgönül and J. Zou, “Cost-of-Pass: An Economic Framework for Evaluating Language Models”, ICLR 2026. arXiv:2504.13359. Full.
60. [C60] S. Kapoor, B. Stroebl, Z. S. Siegel, N. Nadgir and A. Narayanan, “AI Agents That Matter”, 2024. arXiv:2407.01502. Full.
61. [C61] H. Pan, L. Huang, X. Zhou, M. Xing and Y. Wu, “Toward Compiler World Models: Learning Latent Dynamics for Efficient Tensor Program Search”, 2026. arXiv:2606.09312. Full.

Not carried over from the survey, because the primary source was not obtained or did not support the claim: Smalltalk image claims, Darklang hashing, Pantograph and Grove, Codong, Lume, Magpie, Codex (Tedrow), Spec IR, Boruna, d-raft, Sockudo, DBOS, P language compilation to C, UNCOL and ANDF, Thrift and OpenAPI, Pact, Aver, Prove, reqlan, translation validation (Pnueli), SyGuS, Futamura and Turchin, PAR, DéjàVu and Gabel and Su, Simon and Kadane, and the comparative scoreboard.

1. History: seventy years of “say what you want, get the program”

The wish behind this topic is old. What has changed is which parts of it are now cheap. The timeline below lists the main attempts; the text after it says what each one achieved and where it stopped.

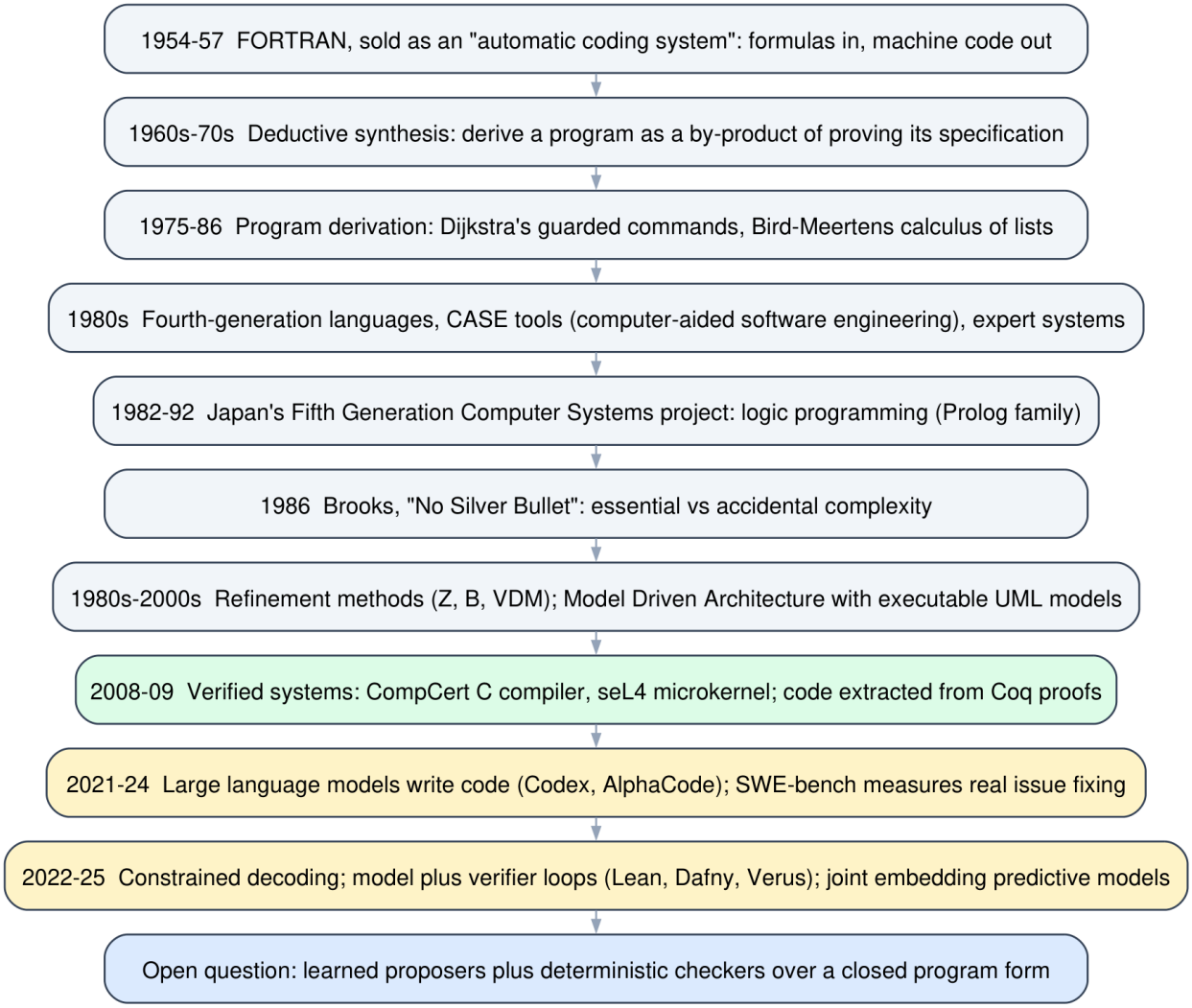

Figure 9. Attempts to turn a statement of intent into a working program. CASE means computer-aided software engineering; UML means Unified Modeling Language; Z, B and VDM (Vienna Development Method) are mathematical specification methods; Coq is a proof assistant. Green: machine-checked systems. Amber: learned models. Blue: the question this topic studies. .

- **Automatic programming, 1950s.** FORTRAN (FORmula TRANslation) was presented as “The FORTRAN Automatic Coding System”. Its authors wrote that its purpose was to let the IBM 704 “code problems for itself and produce as good programs as human coders (but without the errors)” (Backus et al., 1957, p. 188; PDF p. 2). Example: a scientist wrote a formula; the system wrote the machine code. Today that is called a compiler, not automatic programming.
- **Deductive synthesis, 1960s-80s.** Simon’s heuristic compiler (1963) treated programming as heuristic problem solving: a means-end search, built on the General Problem Solver (GPS), that writes routines from descriptions of the state before and after, by “selective trial-and-error search” (Simon, RAND RM-3588-PR, 1963, Summary, p. vii). Manna and Waldinger (1980) treated a program as something extracted from a constructive proof that a specification can be met: “From such a proof, a program to compute z can be extracted” (Manna and Waldinger, STAN-CS-78-690, 1978, Specification section; the same sentence appears in the TOPLAS article, 1980, p. 92); both versions note that “no implementation exists” (report, Motivation, p. 2; TOPLAS, p. 91). The JACM article states Simon’s conclusion directly: “compiling tasks–at least simple compiling tasks–can fruitfully be viewed as problem-solving tasks” (Simon, 1963, Summary, p. 505), and it

makes no practical claim: “no claims will be made for the heuristic programming procedures described here as practical approaches to the construction of compilers” (Introduction, p. 493). Manna and Waldinger observed that theorem proving had been “abandoned as an approach to program synthesis when the development of sufficiently powerful automatic theorem provers appeared to flounder” (TOPLAS, 1980, p. 120). All four texts were read in full.

- **Program derivation, 1975-86.** Dijkstra introduced guarded commands “as a building block” for programs and presented “the formal derivation of programs” from conditions on the final state (Dijkstra, EWD 472, abstract). Bird’s calculus of lists (the Bird-Meertens formalism) is “used to derive efficient solutions” by rewriting with algebraic identities (Bird, 1986, PDF p. 3). Example: for Gries’s problem “to compute the minimum of the sums of all segments” of a list, the direct program takes O(n^3) steps, and rewriting with the calculus yields “a linear time algorithm” (Sec. 4.2, PDF p. 34).
- **Fourth-generation languages, CASE tools and expert systems, 1980s.** Report generators, form builders and diagram tools promised programs from screens and pictures. Brooks reviewed graphical programming, automatic programming and expert systems in 1986 and argued that none of them “promises even one order-of-magnitude improvement” (Brooks, 1986, PDF pp. 1, 8-11). Kemerer reported, citing an earlier study, that “one year after introduction, 70 percent of CASE tools and techniques are never used” and that “the first project written with an integrated CASE tool typically fails to deliver improved results”; he attributes part of the abandonment to a learning curve that dips below the old technology before it rises (Kemerer, 1992, pp. 23-24, Fig. 1).
- **Japan’s Fifth Generation Computer Systems project, 1982-92.** A national effort built on logic programming in the Prolog family; ICOT spent “about 54 billion Yen, that is some 450 million US$” over ten years (van de Riet, 1993, Sec. 1, p. 83). One goal in the 1981 plan was “to lessen the burden of software generation in order that a high level requirements specification is sufficient for automatic processing” (Sec. 3, p. 85). The parallel inference machine met its 100 MLIPS target, but van de Riet scores “break-through in architecture” and “break-through in software” as failures and judges technical impact “small” (Sec. 6, p. 103); on the effort needed to program the system, “there is almost no evidence” (Sec. 4.1, p. 86).
- **Refinement and model-driven methods, 1980s-2000s.** Z, B and VDM refine a mathematical specification step by step toward code. The Model Driven Architecture guide of the Object Management Group “specifies three viewpoints on a system, a computation independent viewpoint, a platform independent viewpoint and a platform specific viewpoint” (OMG, 2003, Sec. 2.2.5, PDF p. 13). Example: draw a class diagram once and generate Java and SQL (Structured Query Language) from it. Hutchinson et al. note that MDE “has been developed largely without the support of empirical data” (2011, Sec. 1, p. 471); among 250+ practitioners surveyed, 74% said MDE “requires them to carry out significant extra training” (Sec. 4.3.1, p. 475), and one interviewee reported 4 to 8 times productivity internally while telling outsiders “1.5 to 2” (Sec. 5.2.6, p. 478). The authors call their results “interesting and suggestive rather than conclusive” (Sec. 3.1, p. 473).
- **Machine-checked systems, 2008-09.** CompCert is a C compiler whose “proof of semantic preservation” was carried out in the Coq proof assistant (Leroy, 2009, abstract, PDF p. 1). seL4 is a microkernel of “8,700 lines of C code” with a machine-checked “proof of functional correctness” (abstract, PDF p. 1); the proof cost “about 20 py” (person-years) (Klein et al., 2009, Sec. 5.2, PDF p. 13). Coq can extract a proved function into ordinary code in “functional languages such as Ocaml, Haskell or Scheme” (Letouzey, 2008, Sec. 1, PDF p. 1); the extraction step itself was not yet mechanically verified, and its correctness rested on paper proofs (Sec. 6, PDF p. 10).

2. Why the earlier attempts stalled

- **Essential versus accidental complexity.** Brooks separated “essential tasks, the fashioning of the complex conceptual structures that compose the abstract software entity”, from “accidental tasks, the representation of these abstract entities in programming languages” (abstract, PDF p. 1). He quoted Parnas: “automatic programming always has been a euphemism for programming with a higher-level language than was presently available to the programmer” (PDF p. 9), and wrote that “even perfect program verification can only establish that a program meets its specification” (PDF p. 11) (Brooks, 1986). Example: a better notation for a payroll rule does not decide what the rule should be.
- **Natural language is ambiguous.** Dijkstra argued that formal texts are, “when you come to think of it, an amazingly effective tool for ruling out all sorts of nonsense that, when we use our native tongues, are almost impossible to avoid” (Dijkstra, EWD 667, para. 6). Example: “sort the users by name and date” has at least

two readings.

- **Specification costs as much as the program.** In seL4 the abstract specification took "about 4 person months" and the proof about 20 person-years, against "a total cost of 2.2 py including the Haskell effort" to build the kernel itself (Klein et al., 2009, Sec. 5.2, PDF pp. 12-13). The 20 py includes about 9 py of research and reusable proof tooling; the seL4-specific proof "was 11 py" (PDF p. 13).
- **Combinatorial search.** Searching all programs grows exponentially with program length. DreamCoder addresses this by learning a library of reusable pieces and a neural network "to guide the search for programs" in a "wake-sleep" loop (Ellis et al., 2021, abstract, PDF p. 1); the paper calls such search "intractable for general-purpose algorithms, because of the combinatorial nature of the search space" (introduction, PDF p. 2); the supplement repeats that finding programs "is difficult because of the infinitely large, combinatorial search landscape" (supplement S4.1, PDF p. 42 of the version with supplement), the same kind of search problem that limited deductive synthesis.
- **Brittleness and adoption.** Diagram-to-code tools generated code that was then hand-edited, and the model and the code drifted apart. Petre reports from earlier surveys that "the biggest perceived problem of model-centric approaches is keeping the model up-to-date with the code (68% agreement)" (Forward and Lethbridge, as reported by Petre, 2013, author copy p. 3); among Petre's own 50 informants, 35 did not use UML (Unified Modeling Language) and 3 used automated code generation (pp. 4-5). Adoption studies of CASE and of model-driven engineering are listed in the ledger as not read, so no figures from them are quoted.
- **Opinion.** The common thread is that each approach fixed one cost (notation, proof, generation) while another stayed manual: writing the specification, searching for the program, or checking the result.

3. Three lanes toward a World of Code

The wish in Section 1 is old. What changed by November 2026 is that several research programmes, working for their own reasons, now put a learned proposer next to something that can say yes or no. This section records three lanes. A lane is a convergence observation: agreement in direction, not a ranking, and not a claim that one programme replaces another. Every figure below is as reported by the cited source. None of them is a measurement of the Reality Code Engine except the final row of Section 3.3.

3.1 Training on popular architectures

This lane keeps a widely used model shape, usually a decoder that predicts the next token (a word piece), and changes the training data or the reward.

Who	As reported	Unfinished in that source
Meta FAIR CodeGen. CWM, a 32 billion parameter dense decoder [B23]	Code is learned better when the model sees execution, not only text. Mid-training uses Python traces and agent trajectories. SWE-bench Verified pass@1 53.9% for the base model, 65.8% with a best of 16 samples at test time. LiveCodeBench 68.6%. CruxEval-O 94.3%. 120 million traced functions. 3 million ForagerAgent trajectories from 10.2 thousand images. An 8 billion parameter ablation: traces lift CruxEval-O from 44.6 to 73.9, and Forager lifts SWE-bench Verified from 18.4 to 22.1.	The authors call trace prediction a "first step". A neural debugger, and uses in verification, are future work in that report.
SWE-RL, Wei et al.	Reinforcement learning on pull-request history. The reward is similarity to an oracle patch, not semantic equality. Llama3-SWE-RL-70B is reported at 41.0% on SWE-bench Verified.	The reward is not semantic equality.
SWE-smith, Yang et al.	Synthetic tasks per repository: 50,000 instances, 128 repositories. SWE-agent-LM-32B at 40.2% pass@1.	Supervised fine-tuning only in the paper. Reinforcement learning is an external system named SkyRL.
DiffuCoder, Gong et al. Mercury, Inception Labs	Non-autoregressive (masked diffusion) code generation. DiffuCoder is a 7 billion parameter model on 130 billion tokens; coupled-GRPO is reported at an additional 4.4% on EvalPlus. Mercury is reported at 1,109 and 737 tokens per second on an H100.	Quality parity with autoregressive models is not established in the sources read here.

Who	As reported	Unfinished in that source
PatchDiff, Wang, Pradel and Liu. METR, Becker et al.	Benchmarks can overstate success, and field productivity can fall. PatchDiff: 7.8% of patches counted correct fail developer tests, and resolve rates are inflated by about 6.4 points. METR, on early-2025 tools: 16 developers, 246 tasks, measured completion time 19% higher. A forecast of 24% lower, and a post-hoc estimate of 20% lower, did not match that measurement.	Both checks are dated to early or mid 2025 tools.
SWE-Bench Pro, Scale AI. Public leaderboard fetched 6 November 2026	Built to resist contamination (copyleft and private repositories) and to ask for long changes: average 107.4 lines across 4.1 files. 1,865 tasks, 41 repositories (731 public, 276 private, 858 held out). Public board that day: Muse Spark 1.1 at 61.50 +/- 3.10; gpt-5.4 (xHigh) at 59.10; Muse Spark at 55.00; claude-opus-4-6 (thinking) at 51.90; gemini-3.1-pro at 46.10. The top four of those ran with mini-swe-agent. Launch text on the same page: GPT-5 at 23.3% and Opus 4.1 at 23.1%, against more than 70% on SWE-bench Verified. Private set: Opus 4.1 at 17.8%, GPT-5 at 14.9%. The page prints no update date.	The oracle is still fail-to-pass and pass-to-pass tests.
Multi-SWE-bench, ByteDance Seed	Issue resolving across Java, TypeScript, JavaScript, Go, Rust, C and C++. 1,632 instances from 2,456 candidates, with 68 annotators.	A November 2026 leaderboard figure was not retrieved. None is stated on this page.

Lane reading. The training reports above move toward semantics: execution traces, environment dynamics, and rewards a checker can see. The test suite is still the oracle, and PatchDiff is evidence that a test suite is incomplete. SWE-Bench Pro's board is higher than its launch text and still grades patches by tests. That is the opening the Laws of Coding name. An exact checker replaces "tests passed" with an accepted Form. Acceptance is part of the world, not a score fitted afterward.

3.2 Creative hybrids

This lane pairs a learned proposer with a gate that already exists: an evaluator, a proof kernel, a type system, or a set of examples.

Who	As reported	Unfinished in that source
AlphaEvolve, Google DeepMind [C52]	A large language model proposes code edits. An evolutionary loop wraps it. Automatic evaluators decide. A 4 by 4 complex matrix multiplication in 48 scalar multiplications. Also datacenter scheduling, circuit simplification, and a speedup of its own training, with no further figure used here.	It works where an evaluator exists.
AlphaProof, Google DeepMind [C38]	Reinforcement learning in the style of AlphaZero, inside Lean. The formal environment grounds the reasoning. 80 million autoformalized statements. At the 2024 International Mathematical Olympiad, 3 of 5 non-geometry problems, a silver-medal result with AlphaGeometry 2, at multi-day compute.	Compute cost. The report is mathematics, not software.
DeepSeek-Prover-V2-671B	Subgoal decomposition by DeepSeek-V3, proofs of the subgoals stitched into a chain-of-thought cold start, then reinforcement learning. The Lean 4 kernel is the gate. miniF2F-test pass ratio 88.9%. PutnamBench 49 of 658. A new ProverBench of 325 problems. 6 of 15 AIME (American Invitational Mathematics Examination) 2024 and 2025 problems in the formal setting, against 8 for DeepSeek-V3 by an informal majority vote.	Mathematics, not software. The formal and informal gap is still nonzero.
Kimina-Prover Preview, Moonshot and Numina	From Qwen2.5-72B. Large-scale reinforcement learning with a formal reasoning pattern in Lean 4. miniF2F 80.7% at pass@8192. Distilled models at 1.5 billion and 7 billion parameters were released.	A preview release.

Who	As reported	Unfinished in that source
Goedel-Prover, Princeton	Natural-language problems autoformalized to Lean (1.64 million statements). Expert iteration builds more than 800,000 proofs. Supervised fine-tuning only. miniF2F 57.6% at pass@32. PutnamBench 7 at pass@512.	No reinforcement learning in version 1.
Vericoding benchmark, Beneficial AI Foundation [C34]	Generate code and a proof from a formal specification. 12,504 specifications. Success 82.2% Dafny, 44.2% Verus, 26.8% Lean. The abstract rounds those shares to 82%, 44% and 27%, which is the rounding in the commitment section below. The split is 3,029 Dafny, 2,334 Verus and 7,141 Lean. Dafny verification on that line of work moved from 68% to 96% in a year. Natural-language descriptions add little.	The cost of writing the specification remains.
Type-constrained decoding, Mündler et al.	The type system constrains token sampling. Authors affiliated with the Swiss Federal Institute of Technology Zurich and the University of California, Berkeley. Compilation errors down 75.3% on HumanEval and 52.1% on MBPP (the short Python set named in that report). A syntax-only constraint gives 9.0% and 4.9% on those two sets.	TypeScript scale only. Types are not full acceptance of behaviour.
DreamCoder, Ellis et al. LILO, Grand et al.	Library learning by compression. LILO adds a large language model and symbolic refactoring (Stitch). DreamCoder is quoted from the full text in Section 2. LILO is abstract-level on this pass.	Small domain-specific languages in the sources read for the lane.
egg, Willsey et al. [C4]	Equality saturation: a set of equivalent programs stored as one e-graph. Principles of Programming Languages, 2021.	Someone still has to write the rewrite rules.
ARC Prize 2024, Chollet, Knoop, Kamradt and Landers. Akyurek et al.	Program synthesis guided by deep learning, plus test-time training. The ARC (Abstraction and Reasoning Corpus) private score on ARC-AGI-1 moved from 33% to 55.5% during 2024. An 85% target was not met. Akyurek et al. update weights at inference on the task's examples: 53.0% on the ARC public validation set with an 8 billion parameter model, and 61.9% when ensembled with program synthesis.	The 85% target was not met. The Akyurek regime is few-shot.
ARC Prize 2025 results page	Winners on ARC-AGI-2 private, at a cost cap of 0.20 dollars per task. NVARC (the winning entry named on that page) 24.0%, with synthetic data, test-time training and a tiny recursive net. ARChitects 16.5%, a two-dimensional masked diffusion language model with recursive refinement. MindsAI 12.6%. Paper awards on that page: a tiny recursive net of about 7 million parameters at about 45% on ARC-AGI-1 and about 8% on ARC-AGI-2; SOAR (evolutionary program synthesis with self-fine-tuning, as that page names it) up to 52% on ARC-AGI-1; CompressARC, a minimum-description-length method with no pretraining, about 20% to 34% on ARC-AGI-1 and about 4% on ARC-AGI-2.	The grand prize was unclaimed. ARC-AGI-3, an interactive track, is in development.
Large Concept Models, Meta FAIR	Autoregressive prediction in a sentence-embedding space, SONAR (the sentence-embedding space named in that report, 200 languages), instead of tokens. Mean squared error, diffusion, and quantized variants. 1.6 billion parameter models on about 1.3 trillion tokens. One architecture scaled to 7 billion parameters on about 2.7 trillion tokens.	In the abstract, a concept is a sentence, a proxy. No code-specific result is given there.

Lane reading. Each of these hybrids has the shape stated in the principles above: the model proposes, and a deterministic gate decides. AlphaEvolve and AlphaProof are clear points of contact. The Lean provers are the same shape with the kernel as the gate, and the gains they report came from data generated against that kernel. The ARC Prize 2025 page shows program search, test-time training, diffusion and small recursive nets on one loop: propose, check against examples, refine. The shared limit is also shared. They need an evaluator, a kernel, or a set of examples. The Laws of Coding are the claim that an environment of that kind exists for coding in general, not only for mathematics and for benchmark kernels. Large Concept Models are the closest token-free relative of JEPA in language: prediction in a concept space, still without an exact acceptance step.

3.3 Reconceptions of computation, programming and coding

This lane does not start from a benchmark score. It starts from a question about what a program is.

Who	As reported	Unfinished in that source
HVM2, Higher Order Company	Interaction combinators as a model of computation, with parallelism inherent in the model. This page cites the model.	Figures in that paper are not used here. Production adoption is open.
Unison [C2]	A definition is the hash of its syntax tree. Names are views over that hash. The language is in production use.	Ecosystem size.
Lean, Dafny and Verus communities, via the vericoding benchmark [C34]	A proof is a program the kernel can check. Correctness is a checkable object. The measured floor used here is the vericoding row above.	The cost of writing the specification remains.
egg [C4]	Program equivalence classes as data. Programs live in a quotient.	Completeness of the rewrite rules.
Lean FRO (Focused Research Organization), Year 1 roadmap, August 2023 to July 2024	Lean as a language and a proof assistant for mathematics and for software and hardware verification. The deliverables named on that page include a standard library, a module system, Lake and Reservoir, proof automation, machine-learning integration and a new code generator.	Later-year roadmaps are not used on this page.
Conal Elliott, denotational design	"The instance's meaning is the meaning's instance." A semantic model defines correctness and carries it to the implementation. The note is from 2009, revised 2016.	A manual design discipline.
Radul and Sussman, propagators	Computation as autonomous machines that add information to shared cells. The account unifies expression-style programming and constraint programming, including incremental distributed search. A 2009 technical report, with a Scheme implementation.	Adoption.
Chollet, on the measure of intelligence [T37]	Intelligence as skill-acquisition efficiency. ARC is the measure. Program synthesis is the framing. The 2019 paper is the citation. Later talks on synthesis done at the moment of the task are not cited on their own. Where the ARC Prize 2024 report names deep-learning-guided program synthesis, that report is the citation.	The formal measure is not operationalized beyond ARC in the abstract read for this lane.
This project, the Laws of Coding	A canonical Form is the object. Surface text is a projection. Cost is counted. Measured results: H1 median rho 9.34 over 185 programs; H2 `C_check / E_model = 0.000582 <= 0.05` (`1,718×` cheaper on M5 Max); H5 & H11 on unseen disjoint Form shapes via `FormEBM-v2` (AUROC `0.8922`, `93.0%` 5-step monotonicity) and span-normalized `FormEBM-v3` (`6.05M` params: AUROC `0.9437`, `94.0%` `E_depth` / `92.0%` `E_total` monotonicity, zero-shot OOD H15 pooled AUROC lifted from `0.2585` to `0.5278`); H7, 1,275 mutants and 0 violations on the fuzzed arm; H8, 10,500 streams, six targets, 0 mismatches; H14 run 3, 111 of 151 (73.5%) closed at 0 model calls; H15 across the 40 residual tasks, 17/40 to 34/40 single-shot, `27/40` (`k = 16`) for `FormJEPA-v1` (`8.42M` params, `474.76 mJ` RAPL core `reported_j`), `40/40` (`0` contradictions) in `H15-Iterative` on `Opus 5.5` and `Sonnet 5.5`, `40/40` 0-call Gear-0 replay (`0.325 µs`, `1.77 µJ`/probe), `16.67×` 3-Gear episode reduction on `Zipf(0.85)` (`200/200` admitted), and 151-task `rce repair` rung `f` audit (`3/119 = 2.5% > 2.0%`, `0` contradictions, stays in `observe`).	Full intent-to-Form synthesis (H3), cross-module signature-and-trace OOD energy calibration beyond `0.5278`, H4, H6, H9, H10, H12, H13, the diffusion and evolutionary arms of H15, and wall-meter `measured_j` remain open.

Lane reading. Unison (identity is a hash of structure), egg (programs as equivalence classes), interaction nets (computation as local rewriting) and proof assistants (correctness as an object) each hold one piece of what the Laws of Coding state together: one canonical Form, exact replay, projection to targets, and checkable acceptance. Elliott's denotational design and the propagator account are earlier statements of the same demand: meaning defines

the implementation, and computation adds information rather than only emitting text. They are independent convergence, each from a different starting point. Chollet's measure says the thing to count is how efficiently a system acquires a skill, which is why proposer comparisons track both call budgets and physical energy (Section 8).

4. Why now: the case for closed-form, deterministic, AI-proposed code

This section is opinion, built from the history above and from the three lanes in Section 3. It is a case to be tested, not a result.

- **Checkers are cheap and trustworthy where the earlier attempts lacked them.** Using Csmith, a random program generator, the authors found that CompCert was "the only compiler we have tested for which Csmith cannot find wrong-code errors" after "about six CPU-years" of trying (Yang et al., 2011, Sec. 3.1, PDF p. 6). Machine-checked rules can carry the trust that a generator cannot. CompCert itself uses this pattern: its register allocator runs an unverified graph-coloring heuristic and checks each result with a verifier proved correct in Coq, because graph coloring is "easier to validate a posteriori than to directly prove correct" (Leroy, 2009, Sec. 4.2, PDF p. 7).
- **Deterministic semantics answer the nondeterminism problem.** If the generator is allowed to vary but the accepted artifact is checked and replayed exactly, run-to-run variation in the model stops mattering for the result.
- **Verifier-in-the-loop evidence exists.** AlphaCode's behavioral filtering, Codex-S's oracle test selection (77.5% versus 44.5% for log-probability ranking), Clover, AutoVerus and AlphaProof all show large gains when a check selects among proposals. Section 3 records the same shape in later systems: a model proposes, and an evaluator, a proof kernel or a type system decides.
- **Learned predictors now work in latent space.** I-JEPA and V-JEPA 2 show that predicting a summary of the target, rather than every detail, can work well for those tasks: V-JEPA 2 had higher success on 4 of 5 robot tasks (tied on one) and planned about 15x faster than a generative video model in its comparison (Table 3, PDF p. 15). In a program, details such as identifier spelling, formatting and target language can be filled in deterministically after the structure is chosen. Section 5 states the premise and where it stops once Forms and the Laws of Coding exist.
- **Token-by-token generation is expensive.** LLaMA 65B, sharded across 8 to 32 V100 GPUs with a 512-token output limit, used "about 3-4 Joules for a output token", counting GPU energy only (Samsi et al., 2023, Secs. III-C and IV-C, PDF pp. 4-6). In a study of 88 models, text-generation models averaged 0.047 kWh (kilowatt-hours) per 1,000 queries, each query generating only 10 new tokens (Table 2), and "multi-purpose, generative architectures are orders of magnitude more expensive than task-specific systems", partly because a generator chooses among "an entire vocabulary" (Luccioni et al., 2024, abstract, PDF p. 1; Sec. 4.1 and Table 2, PDF p. 6; Sec. 4.2.1, PDF p. 8).
- **Same-host check-to-model comparison (Section 8, H2).** Rather than mixing external V100 numbers with local laptop timings, Section 8 measures both on the same Apple M5 Max host: `0.04904 J` per Form check versus `84.25 J` per candidate on local `Qwen 3.6 35B-A3B` (`124.81 J` per admitted residual repair).

What could still fail

- **Expressiveness ceiling (H6).** If real programs need effects or nondeterminism the Form cannot state, the "all code" premise fails.
- **Intent may be the bottleneck.** A smaller output space does not fix underspecified intent. Signal: H1 and H2 hold while H3 fails.
- **Spec gaming.** The checker proves a Form is well-formed and meets stated properties, not that it matches what the user meant. Benchmark tasks carry behavioral traces checked by replay; the residual risk is labeled, not solved.
- **Corpus scarcity.** Signal: H3 flat as the corpus grows.
- **Collapse.** JEPAs need anti-collapse measures, such as the EMA (exponential moving average) target encoder of I-JEPA and V-JEPA 2; LeCun (2022) proposes VICReg-style information maximization instead (Sec. 4.5). A collapsed Form embedding makes H5 fail. Section 5 states this limit beside the others.
- **Measurement error.** Rootless counters (macmon `IOReport` L2, kernel `ri_energy_nj` L3, RAPL) cover SoC or process counters (`reported_j`), not alternating-current (AC) wall power (`measured_j`).

- **Baseline drift.** Co-tenant processes on shared hosts drift L2 idle baselines (Falsifier 6), requiring per-process-identifier (PID) L3 counters or dedicated idle hosts.

5. JEPA as premise, and where it stops

The premise stays. A JEPA predicts in an embedding space, not in the space of the raw input. The thesis cites LeCun's 2022 statement of that idea. V-JEPA 2 reports 62 hours of robot video and zero-shot pick-and-place success from 65% to 80% [B1]. The action-conditioned predictor, as the proposer section states from the same report, was trained on fewer than 62 hours of unlabeled robot video. LLM-JEPA reports that an embedding-space loss beats a token loss on a natural-language-to-regular-expression task, on GSM8K (a grade-school math set) and on Spider (a text-to-database query set), at 1 billion parameter scale (Huang, LeCun and Balestriero, 2025). For code, this matches the intuition that the useful object is meaning, not tokens.

Once Forms exist, and once the Laws of Coding fix equivalence and acceptance, that premise is not the whole method. Six limits follow. They are reasons to widen the proposer, not reasons to drop the premise.

1. **An approximate geometry, where an exact one can be computed.** JEPA learns an approximate latent geometry. The Laws of Coding give an exact one: the canonical Form and the equivalences the checker already decides. Learning those by self-supervision spends data on a map that can be computed. H10 is the test of whether learning on the quotient saves data. H10 has not been run.
2. **Collapse.** Representation collapse, and the need for regularizers against it, are open problems in JEPA training. An exact canonicalizer cannot collapse. It maps each surface program to one Form by a rule, not by a loss.
3. **No native acceptance.** JEPA has no native decoder and no acceptance step. Code needs a discrete, checkable output. The energy score ranks candidates. It does not decide. The checker decides.
4. **Points, not a quotient.** Alternate Forms are a quotient. Many surface programs can be one Form, and one Form can be projected to many targets. JEPA embeds points. It does not certify that two points are the same Form. An e-graph does. A hash of structure does. In Section 8, `rce fmt nameless` maps `7` of `9` alpha-renamed local-variable mutants onto the identical memory-mapped Gear-0 (`DNULLMM1`) slot at zero model calls.
5. **Scale of the evidence.** LLM-JEPA's evidence is at 1 billion to 2 billion parameter scale, on small datasets. No source read for this page shows JEPA training for repository-scale code.
6. **The cost of learned planning.** V-JEPA 2's model predictive control (MPC) took about 16 seconds per action [B1]. Search over Forms with a cheap exact checker (`7.32 ms`, `49.04 mJ` L3 `reported_j` on M5 Max) and Gear-0 lookup (`0.325 µs`, `1.77 µJ`) bounds repeated rollout cost before any learned planner is trained.

Consequence. JEPA stays as the premise: predict meaning, not text. The proposer slot generalizes to any learned proposer over Forms. The instances named in Section 3 are autoregressive models with execution mid-training (CWM), diffusion code models, reinforcement learning in a formal environment (AlphaProof and the Lean provers), evolutionary search (AlphaEvolve), library learning (DreamCoder and LILO), and constrained decoding. Large Concept Models sit beside the premise in language, not yet in code. The gate is the same for all of them. Section 6 is the taxonomy. Section 7 says what the world is that they propose into.

6. The proposer slot is architecture-neutral

Figure 1 draws one pipeline. The learned box is a slot, not a single architecture. JEPA is the instance in the figure. The gate drawn outside the box does not change if a different proposer is plugged in. The table says, for each kind of proposer read in Section 3, what it learns, what it needs from the Laws of Coding (a canonicalizer, a checker, a cost meter), the failure mode the sources already name, and which hypothesis it moves or would move. Section 8 now reports both the first purpose-trained latent Form JEPA repair proposer (`FormJEPA-v1`, `8.42M` trainable parameters) and six off-the-shelf autoregressive, reasoning, and hybrid Gated-DeltaNet / MoE models on the 40 residual tasks.

Proposer	What it learns	What it needs from the checker	Known failure mode	Hypothesis it moves / would move
JEPA, an embedding predictor (`FormJEPA-v1` in Section 8)	An approximate latent geometry of Form structure and edits (EMA target encoder + VICReg + latent-conditioned typed edit decoder)	A canonicalizer, a checker, a cost meter, and a discrete edit head. The energy score does not accept	No native acceptance. Uncalibrated latent distance alone gives AUROC `0.618`, and compact non-pretrained models miss open multi-module API migrations (`0/9` on H15) without Gear-2 fallback	`FormJEPA-v1` measured on H15 (`22/40` at `k = 1`, `27/40` at `k = 16`) and disjoint synthetic defects (`177/200`); full intent-to-Form H3, calibrated H5/H11, H10, and H13 remain open
Autoregressive / reasoning / trace-trained decoders (Claude, Grok, generative pretrained transformer open-weights (GPT-OSS), DeepSeek, Qwen 3.6, CWM class)	Next-token prediction, chain-of-thought reasoning, hybrid Gated-DeltaNet / MoE, or execution traces	A checker as acceptance, plus Gear-0 crystallization so recurring shapes do not re-invoke the model	Token-space retries spend `84.25 J` per local candidate (`Qwen 3.6`), and long reasoning traces can exhaust fixed completion caps (`DeepSeek V4.1 17/40`)	First single-shot (`k = 1`) H15 arms measured in Section 8 (`17/40` to `34/40`); RJ-F6 tests whether trace-trained token models match Form-quotient training at equal energy
Diffusion code model	A masked or denoised program, not a left-to-right token stream	A checker at acceptance. A constraint can also act while sampling	Quality parity with autoregressive models is not established in the sources read here	The diffusion arm of H15 (open)
Reinforcement learning in a formal environment	Steps that a kernel accepts, as in AlphaProof and the Lean provers of Section 3	The kernel is the environment. For software, that role is the checker	In the reports read here the domain is mathematics, not software. Compute cost is part of the AlphaProof result. Goedel-Prover version 1 has no reinforcement learning	H3 and H12, if the same loop is run on Forms (open)
Evolutionary search	Edits, kept or dropped by an evaluator, as in AlphaEvolve and in SOAR on the ARC Prize 2025 page	An evaluator. Here the evaluator is the checker, not a test suite alone	The loop works where an evaluator exists. Example checks, as on ARC, are not a general acceptance rule for software	The evolutionary arm of H15, on the residual of H14 (open)
Library learning (DreamCoder, LILO)	A reusable library, by compression, with or without a large language model in the loop	A checker on the programs the library emits	The domains in the sources read for this lane are small	H10 (open)
Constrained decoding	Nothing new in the weights. The constraint masks illegal tokens, as in the type-constrained report	A grammar or a type system at sampling time, and the checker for behaviour	The headline result is TypeScript scale. Syntax and types are not full acceptance. Compilation errors fell 75.3% on HumanEval and 52.1% on MBPP, against 9.0% and 4.9% for syntax only	H3, only for the share of failures that are type failures
Large Concept Models	The next sentence embedding, not the next token	An exact acceptance step, which the abstract does not have	A concept there is a sentence. No code-specific result in the abstract	None of H1 to H15 until a code result exists

The deterministic ladder in the proposer section runs before any of these (H14: `111/151` closed at `0` calls, leaving `40` residual tasks). Section 8 reports both the six-model off-the-shelf H15 baseline plus Shape D (`DNULLMM1`) Gear-0 crystallization and the purpose-trained `FormJEPA-v1` arm plus 3-Gear Cascade on those 40 tasks, while keeping the diffusion and evolutionary arms, intent-to-Form H3, and wall-meter server joule parity explicitly open.

7. A World of Code

Definition. A World of Code is the state space of Forms, plus the exact transition and the exact acceptance rule. In the notation of the thesis, the state space is S, the transition is the function δ, and acceptance is the checker's verdict on a Form. "World model", for code, means a model of this space. Unlike a physical world model, the physics is known exactly. Part 4 of the thesis says that for a JEPA-style predictor. The same physics bounds every

proposer in Section 6. CWM learns an interpreter from traces. V-JEPA 2 learns physics from video. In a Form, the transition is given. There is no irreducible noise in the next state.

Where learning is removed, and where it remains. Canonicalization, equivalence and replay are computations. Once the Laws of Coding fix them, a model does not have to learn them. What remains to learn is the map from a stated intent to a Form, guidance of the search, and ranking of whatever the deterministic ladder leaves open (`40` real-history tasks after H14 run 3). Once any proposer solves one of those residual tasks and passes the gate, Gear-0 crystallization turns every recurrence (and `7/9` alpha-renamings under `rce fmt nameless`) into a `0.325 µs` deterministic lookup.

Evaluation. A test suite is not the Laws of Coding. PatchDiff reports that 7.8% of patches counted correct fail developer tests, and that resolve rates are inflated by about 6.4 points. The METR trial reports 16 developers, 246 tasks, and a measured completion time 19% higher with early-2025 tools. SWE-Bench Pro, on the board fetched 6 November 2026, still uses fail-to-pass and pass-to-pass tests as the oracle. Multi-SWE-bench widens the set of languages to Java, TypeScript, JavaScript, Go, Rust, C and C++. Acceptance on this page is the gate (`rce check` exit `0` plus obligation preservation), verified further against historical originals by `rce hash` and `rce equiv`.

Falsifier of the widened claim. RJ-F6, stated with the others in Section 8, names the observation that would defeat the Form quotient as a help to proposers: if a token-space model closes the residual of 40 tasks at equal energy and equal gate acceptance without the Form quotient. Single-shot frontier token models close up to `34/40` (`85.0%`) on first encounter, whereas `FormJEPA-v1` (`8.42M` parameters over quotient-normalized Form S-expressions) matches local 35B token-space repair at `27/40` (`k = 16`) at over an order of magnitude lower `reported_j` and `rce fmt nameless` supplies `7/9` zero-call quotient hits under alpha-renaming; controlled equal-size quotient-versus-surface sample-efficiency tests (H10) and intent-to-Form H3 remain open.

8. One concrete instance: the Reality Code Engine

RCE is one closed program form among possible others; the questions above apply to any language with a small, checkable program shape. In RCE every program is a Form P = (S, E, δ, D, λ):

- **S, state:** the typed data the program holds, for example the cursor position and text buffer of an editor.
- **E, events:** the inputs it can receive, for example "key pressed" or "file opened".
- δ **(delta), transition function:** the rule giving the next state from the current state and an event.
- **D, derivations:** the recorded, hash-chained history of steps, which makes every run replayable bit for bit.
- λ **(lambda), output:** what the program shows or emits for a given state, for example the visible lines of text.

A checker accepts a Form or names the rule it breaks. At commit `1762fb4b` (20 November 2026), the public reference checker replays an accepted Form exactly, passes `2,221` of `2,236` conformance tests with `0` failures (`837` Rust unit tests), emits Rust, Python, JavaScript and Scala, and is backed by `5,537` Lean 4 kernel theorem constants across `64` modules (theorems T1 to T63). On Apple M5 Max, dual-channel `reported_j` receipts record `49.04 mJ` per Form check (L3 `ri_energy_nj`, `7.32 ms`/Form) and `15.44 J` per full 62-Form check, trace-run and five-target emit suite (`2.32 s`/pass; `measured_j = None`, `board_metered = false`). The full description is on the RCE topic page and at github.com/openIE-dev/rce.

RCE-JEPA is the working name, kept so the thread stays findable. The subject is World Coding. JEPA, a joint embedding predictor, is the premise: predict meaning, not surface text. The proposer slot is any learned proposer over Forms. Checking, proofs, derivation, emission and placement stay with RCE. The aim is the best performance per clock cycle and per joule. *Working hypothesis:* if all code can be encoded in the Form, all code can be produced by joint embedding or energy-based models over it. The hypotheses below are designed so that the evidence can refute this.

Defined quantities

- **Output space.** Free-form generation over vocabulary V with length L has raw space $|V|^L$. Ω_n (omega) is the set of valid Forms of size n (Form nodes); $\Omega_n{}^{\checkmark} \subseteq \Omega_n$ is the subset the checker accepts. Reported as $\log_2|\Omega_n|$ bits, estimated by grammar counting and sampling.
- **Description length.** b(P) is the bits to encode Form P canonically; b(code) = tokens $\times$ $\log_2|V|$ for the equivalent emitted source. Ratio ρ (rho) = b(code) / b(P).

- **Verification cost.** $C_{check}(P)$: wall time, cycles and joules for the checker to decide P. Unit tests give C_{test}; a passing test is evidence, not proof.
- **Acceptance probability.** a = P(checker accepts and task specification satisfied | intent), measured as pass@1 over the benchmark.
- **Joules per accepted program.** $J_{acc} = (E_{model} + E_{check} + E_{retry}) / N_{accepted}$, summed over a benchmark run (labelled `reported_j` for L2/L3 counters and `measured_j` only when an external hardware power meter is attached).
- **Cycles per accepted program.** K_{acc}, the same in CPU (central processing unit) or GPU cycles where counters are readable.
- **AUROC** (area under the receiver operating characteristic curve): the probability that a randomly chosen accepted Form gets a lower energy than a randomly chosen one that requires change; 0.5 is chance, 1.0 is perfect.

Argument (opinion, to be tested). A Form has five typed slots with fixed composition rules. Constraining output to Ω_n removes most of $|V|^L$ before the model is consulted, so fewer bits are predicted per program ($\rho > 1$), the checker gives a definite verdict, accepted or requires change, in place of a probabilistic test, and a requires-change verdict is cheap and structured, naming the rule, so retries are guided rather than blind.

Falsifiable hypotheses H1 to H15

All fifteen hypotheses in numerical order. H1 to H7 test the RCE-JEPA thesis directly, H8 to H10 test the "Laws of Coding" thesis above, H11 to H13 test the proposer, H14 measures the deterministic ladder beneath it, and H15 compares proposer classes on the 40-task residual alongside Shape D (`DNULLMM1`) Gear-0 crystallization.

ID	Hypothesis	Metric	Supported if	Disproved if
H1	Form space is much smaller than source space for the same programs. **Supported on hand-written corpus** (median $\rho = 9.34$ over 185 programs)	ρ = b(code)/b(P), median over corpus	$\rho \geq 5$	$\rho < 2$
H2	Checker is cheap relative to generation. **Supported in `reported_j` on Apple M5 Max**: `C_check / E_model = 0.04904 J / 84.25 J = 0.000582` (`1,718×` cheaper; `measured_j = None`)	C_{check} / E_{model} per candidate (joules)	≤ 0.05	≥ 0.5
H3	Representation-space prediction reaches useful acceptance	pass@1 accepted Forms on held-out intent-to-Form tasks	$\geq$ baseline pass@1 on the same tasks	$< 0.5 \times$ baseline
H4	Energy per accepted program falls	J_{acc}(RCE-JEPA) / J_{acc}(baseline)	≤ 0.1	≥ 1
H5	Energy ranking concentrates mass on valid Forms. **AUROC threshold met on unseen test Form shapes**: contrastive `FormEBM-v2` (`5.85M` params) reaches AUROC `0.8922 >= 0.85` (`84.05%` solvable top-1 accuracy), and span-normalized decomposed `FormEBM-v3` (`6.05M` params) raises unseen-shape AUROC to `0.9437 >= 0.85` (`0.9386` on `E_def` alone) across `3,064` candidates on `200` unseen test beams (vs `0.618` for uncalibrated `FormJEPA-v1`), with `83.44%` `E_def` and `80.98%` `E_total` solvable top-1 accuracy (`68.0%` / `66.0%` raw across all `200` beams) and `2.23×` / `2.14×` fewer `rce check` calls; zero-shot OOD H15 pooled AUROC rises from `0.2585` to `0.5278` (`0.6018` on large-span valid vs short invalid), while cross-module signature/trace OOD calibration remains open	top-1 type-check rate; AUROC of E for accepted vs requires change	≥ 0.8 top-1 valid; AUROC ≥ 0.85	AUROC ≤ 0.6
H6	The reference apps are encodable	share of the 6 reference apps plus RCEdb fully expressed as Forms that pass the checker and replay bit-identically	7 of 7	any app requires an escape hatch outside the Form

ID	Hypothesis	Metric	Supported if	Disproved if
H7	Security is model-independent (fuzzed arm: 0 violations in 1,275 mutants; model arm open)	adversarial Forms (fuzzed and model-generated) accepted by the checker that violate a stated invariant	0	≥ 1 confirmed
H8	Determinism: every accepted Form replays to a bit-identical trajectory and output. **Supported** (0 mismatches on 10,500 streams across 6 targets)	mismatches over at least 10,000 recorded event streams on at least 3 different machines and operating systems	0	a single mismatch
H9	Cost is predicted before execution	median error of the cost-table prediction against measured energy (J, named meter) for a stated machine, over the test families	$\leq 20\%$	$> 50\%$
H10	Learning on the quotient saves data	training examples a predictor of canonical Forms needs to reach the same test acceptance rate as one of equal size trained on surface code	at most half (a saving of 2 times or more)	a saving below 1.25 times
H11	Energy is monotone in edit distance. **Supported on unseen 5-step cumulative defect chains**: contrastive `FormEBM-v2` reaches `93.0% >= 80%` strict 5-step monotonicity (Spearman ρ `= 0.9931`), and two-head `FormEBM-v3` reaches `94.0% >= 80%` on `E_depth` (`94/100`, Spearman ρ `= 0.9966`) and `92.0% >= 80%` on composite `E_total` (`92/100`, Spearman ρ `= 0.9846`, vs `36.0%` for uncalibrated `FormJEPA-v1`); unconstrained same-line random-walk slices reach `38.0%` (`v2`) and `32.0%` / `18.0%` (`v3` `E_depth` / `E_total`)	Share of test families in which energy E rises with edit distance 1 to k from an accepted fit	$\geq 80\%$ of families	$< 80\%$ of families
H12	The decision head is calibrated on gate outcomes	ECE per decision type, after 10^4 receipts	ECE ≤ 0.1 and below a frequency table over shape hashes	ECE > 0.2, or no better than the frequency table
H13	A discrete latent plans better than a continuous one over long edit sequences	Planning success at edit-sequence lengths 1, 5 and 20, discrete arm against continuous SIGReg arm	Discrete arm higher at length 20	Neither arm reaches the H3 threshold
H14	A deterministic ladder closes a share of tasks with no model call. **Supported by its registered rule**: three registered runs [B30, B31, B32], reported below the table	Share of a registered task set that the public deterministic ladder (reuse, lookup by type, determined rewrites, bounded enumeration) closes with zero model calls, every closure admitted by the checker, at a frozen ladder commit	$\geq 10\%$ of the registered tasks, reported with the rung that closed each (registered threshold, unchanged). Measured: run 1, 118 of 120 (98.3%), on mutations the ladder inverts by construction; run 2, 31 of 151 (20.5%), 27 of them guard negations; run 3, plain arm, 111 of 151 (73.5%), 110 of them guard negations and renames	$< 10\%$ of the registered tasks, the same floor as falsifier RCE-F1 of the RCE paper. Not met in any run. On the 41 tasks from real history, which the rule does not single out, runs 2 and 3 each close 1 of 41 (2.4%)

ID	Hypothesis	Metric	Supported if	Disproved if
H15	Proposer classes differ on the 40-task residual. **Measured across single-shot proposers [B33], FormJEPA-v1 [B34], bounded checker-feedback H15-Iterative + FormEBM-v2 [B35], and span-normalized FormEBM-v3 [B36]**: single-shot models admit 17/40 to 34/40, FormJEPA-v1 admits 22/40 (k = 1; 25/40 with FormEBM-v2, 24/40 typed / 27/40 multi-proposer with FormEBM-v3 E_def) and 27/40 (k = 16), H15-Iterative closes 40/40 (100.0%) on Opus 5.5 and Sonnet 5.5 with 0 contradictions (39/40 on Grok 4.20), warm DNULLMM1 Gear-0 replays 40/40 **at 0 calls**, and the 3-Gear Cascade admits 200/200 Zipf(0.85) arrivals with 16.67× fewer Gear-2 episodes; diffusion, evolutionary, and wall-meter arms remain open	Gate-accepted share (C(z) = 1) of the 40 residual tasks from the plain arm of H14 run 3, audited by rce check, obligation preservation, rce hash, and rce equiv	Registered runs report per-arm shares and local reported_j [B33, B34, B35, B36]; diffusion/evolutionary arms and server wall-meter parity remain open	Classes scored on different tasks or unverified outputs

H14, measured. The ladder is `rce repair` of openIE-dev/rce, with the rungs of the proposer section above and an enumeration bound of 2000 candidates. It has three runs, each registered before it ran, and together they are the current result. Run 1 [B30] closed 118 of 120 tasks with zero model calls, but 119 of its 120 tasks were operator and integer-literal mutations, which bounded enumeration inverts by construction, and 9 of the 118 closures pass the examples and checks without being the original program. Run 2 [B31] has no operator or literal mutation: 41 changes from the git history of openIE-dev/rce, 27 guard negations and 83 renames, 151 tasks in all. It closed 31 of them (20.5%) with zero model calls: 1 of the 41 history tasks; all 27 guard negations, every one by the enumeration edit that removes a `not`, which inverts that mutation by construction; and 3 of the 83 renames, none of them restored to the original. Of the 41 history tasks, 39 are programs updated because the language changed, not defects; only 2 are defects of a program, and the ladder closed one of them. Without the guard negations, 4 of 124 tasks close (3.2%). Run 2's prediction P3, that 10% to 40% of the renames close, failed, and its falsifier F1 was met: the closure rate of run 1 does not carry over to changes from real history. Run 3 [B32] ran run 2's 151 tasks unchanged, at openIE-dev/rce `8a5d90ea`, with rung 5, renames, after rung 4; its registration records that renames are the class rung 5 was built for after run 2. Its plain arm, the ladder given only the task, closed 111 of 151 (73.5%) with zero model calls, 103 of them restored to the original: all 83 renames (80 at rung 5, 3 at rung 4), 77 of them restored; all 27 guard negations, each with its outcome in run 2; and 1 of the 41 history tasks, the same one as in run 2. Without the guard negations, 84 of 124 tasks close (67.7%). Its known-good arm gave the ladder each task's original program as well: it closed 116 of 151, 114 restored, including all 83 renames restored and 6 of the 41 history tasks. Run 3's prediction P6, that in the known-good arm every guard and history task keeps its plain-arm outcome, failed: 5 more history tasks close, and the one closed in the plain arm closes at rung 5 instead of rung 3. None of run 3's six falsifiers was met, and the ladder reported zero model calls in both arms. H14's registered rule is support at 10% or more of the registered tasks, and falsification below 10%, for the share the public deterministic ladder closes with zero model calls, every closure admitted by the checker, at a frozen ladder commit. The plain arm meets those conditions. By the rule, H14 is supported: the plain arm of run 3 closes 111 of 151 (73.5%), leaving 40 open residual tasks from git history.

H2, single-shot H15, and Shape D (`DNULLMM1`) Gear-0 on the 40 open residual tasks, measured [B33]. Registered prior to execution (`rce-jepa-h15-and-shape-d-gear0-2026-11-22`, `sha256: b64c4c671ae85755...`) against `openIE-dev/rce` commit `1762fb4b` (`2026-11-20`, binary `sha256: 1d3b0d07ff5b2e8f...`), the 40 residual tasks (`sha256: 3ca3a32769145e9b...`; 25 `W-SYN-1`, 5 `W-LEN-1`, 4 `W-NAME-1,W-TY-1`, 3 `W-NAME-1`, 1 `W-WORLD-1`, 1 `W-CAP-3,W-WORLD-1`, 1 `W-INT-1`) were evaluated under strict gate predicate `C(z) = 1`: (1) `rce check candidate.rce` exits `0` (enforcing `W-UI-1` and `W-MUT-1` non-vacuity), (2) `no_spec_deletion` preserves all declared `state`, `event`, `error`, `struct`, `kind`, `fn`, `extern`, and `step` headers plus every `example` left-hand-side trace and `check` count of the mutant, and (3) secondary equivalence audit via `rce hash` (`restored`), `rce equiv fuel 2000` (`behaviour-equal`), and `rce repair` rung `f observe` mode.

Arm / Architecture Family	Model	Calls	Admitted (C(z)=1)	Restored + Beh-Eq	In / Out Tokens	Cost ($)	On-Chip reported_j
Arm 0: Shape D Cold Static Gear-0	DNULLMM1 (0 records) + a..e	0	0 / 40 (0.0%)	0	0 / 0	$0.0000	< 0.001 J (37.7 µs)
Arm 1a: Cloud Autoregressive + Adaptive Thinking	claude-opus-5-5	40	**34 / 40 (85.0%)**	33 (30 + 3)	114,855 / 32,159	$1.3783	not_available (N)
Arm 1b: Cloud Autoregressive + Adaptive Thinking	claude-sonnet-5-5	40	**32 / 40 (80.0%)**	32 (30 + 2)	114,855 / 30,764	$0.8060	not_available (N)
Arm 2c: Cloud Reasoning Transformer	xai:grok-4.20-0309-reasoning	40	**31 / 40 (77.5%)**	30 (28 + 2)	76,546 / 18,474	$0.2747	not_available (N)
Arm 2a: Cloud Reasoning MoE (Cerebras)	cerebras:gpt-oss-120b	40	**29 / 40 (72.5%)**	27 (25 + 2)	78,825 / 43,596	$0.1340	not_available (N)
Arm 3: Local Hybrid Gated-DeltaNet + Sparse MoE	Qwen3.6-35B-A3B (quantized Q5_K_M)	40	**27 / 40 (67.5%)**	26 (25 + 1)	82,330 / 18,411	$0.0000	**3,369.89 J** system-under-test (124.81 J/adm)
Arm 2b: Cloud multi-head latent attention (MLA) Sparse MoE (Fireworks)	deepseek-v4p1-flash	40	**17 / 40 (42.5%)**	16 (15 + 1)	63,446 / 64,215	$0.0930	not_available (N)
Arm 5a: Purpose-Trained Latent Form JEPA (k = 1) [B34]	FormJEPA-v1 (8.42M params, greedy)	40	**22 / 40 (55.0%)**	22 (20 + 2)	Form S-expr	$0.0000	**58.65 mJ** core / **1.19 J** pkg RAPL/prop
Arm 5b: Purpose-Trained Latent Form JEPA (k = 16) [B34]	FormJEPA-v1 (8.42M params, latent beam)	40	**27 / 40 (67.5%)**	27 (25 + 2)	Form S-expr	$0.0000	**474.76 mJ** core / **6.74 J** pkg RAPL/prop
Arm 6: 3-Gear Cascade v1 (t = 1 Cold Gear-0 → FormJEPA-v1 → Single-Shot Opus 5.5) [B34]	DNULLMM1 + FormJEPA-v1 + Opus 5.5	13 G2	**34 / 40 (85.0%)**	33	Form + 13 cloud	**$0.8108**	FormJEPA RAPL + 13 cloud (N)
Arm 4a: Shape D Warm Gear-0 v1 (t = 2 Replay) [B33]	DNULLMM1 (34 crystallized)	**0**	**34 / 40 (85.0%)**	33	0 / 0	**$0.0000**	**0.325 µs / 1.77 µJ** probe + rce check
Arm 7a: Bounded Checker-Feedback + Stdlib Headers (<= 5 turns) [B35]	claude-opus-5-5 (H15-Iterative)	42	**40 / 40 (100.0%)**	39 (34 + 5)	133,527 / 45,711	$1.8104	not_available (N)
Arm 7b: Bounded Checker-Feedback + Stdlib Headers (<= 5 turns) [B35]	claude-sonnet-5-5 (H15-Iterative)	43	**40 / 40 (100.0%)**	39 (35 + 4)	137,385 / 46,668	**$1.1122**	not_available (N)
Arm 7c: Bounded Checker-Feedback + Stdlib Headers (<= 5 turns) [B35]	xai:grok-4.20-0309-reasoning (H15-Iterative)	50	**39 / 40 (97.5%)**	37 (30 + 7)	269,930 / 28,903	$4.0555	not_available (N)
Arm 4b: Shape D Warm Gear-0 v2 (t = 2 Full Replay) [B35]	DNULLMM1 (40 crystallized)	**0**	**40 / 40 (100.0%)**	39 (34 + 5)	0 / 0	**$0.0000**	**0.325 µs / 1.77 µJ** probe + rce check

- **Hypothesis H2 supported (1,718× local check-to-model energy advantage):** On Apple M5 Max, per-Form checking costs C_check = 0.049038 J (L3 ri_energy_nj, 7.32 ms) while one local Qwen 3.6 35B-A3B candidate generation costs E_model = 84.247 J (3,303.62 J L2 GPU + 66.26 J L3 llama-server

PID across 40 candidates; 14,476.18 J gross L2 SoC, with derived_marginal_j invalidated by Falsifier 6 at 68.31% idle drift), yielding C_check / E_model = 0.000582 <= 0.05 (measured_j = None, board_metered = false).

- **Shape D (DNULLMM1) Gear-0 latency, energy, nameless invariance, and Zipf(0.85) amortization:** Over 40,000 portable operating-system interface (POSIX) mmap lookups against the frozen 34-record store (sha256: d92a57b40de0b0ec..., verified byte-identical pre/post replay), mean probe latency is **0.3246 µs** (324.6 ns, 9,516.5 instructions) and **1.771 µJ** (1,771.3 nJ L3 ri_energy_nj). Keying slots by rce fmt nameless (theorem T7) yields **7 / 9 (77.8%)** zero-call hits when local variables are alpha-renamed. On a 200-arrival Zipf(s = 0.85) recurring stream under verified-only crystallization, 2-Gear DNULLMM1 cuts model calls from 200 to **48 (4.17× reduction)** for Opus 5.5 ($6.81 to $2.63; 5.41× / 37 calls with miss deduplication), 56 (3.57×) for Sonnet 5.5, 57 (3.51×) for Grok 4.20, 66 (3.03×) for GPT-OSS-120B, and 94 (2.13× calls, **1.56× local energy reduction**: 16,457.9 J to 10,553.5 J) for local Qwen 3.6.
- **Scope disclosure on H15:** Two of the 40 historical originals predate the W-MUT-1 vacuous-example rule (spec/examples/57_float_vectors.rce@b2b932da and programs/multi/washer.rce@2cc82de7), accounting for part of the ceiling near 34/40. Diffusion code models, evolutionary Form search, and hardware wall-meter measured_j remain open.

FormJEPA-v1 latent Form repair proposer, 3-Gear Cascade, and 151-task rce repair rung f promotion audit, measured [B34]. Pre-registered before dataset generation, weight initialization, and rung-f execution (rce-jepa-form-jepa-v1-2026-11-22, sha256: 22682e6e936ec120...; rce-repair-rung-f-wave3-eval-2026-11-22, sha256: 04aff4b2b44cdbea...) against openIE-dev/rce commit 1762fb4b (Linux x86_64 binary sha256: 5ef9931ff23a310f...), FormJEPA-v1 (8,424,073 trainable parameters, 13,917,577 with EMA target buffers; checkpoint sha256: cbfbf2274175c737...) was trained from random initialization (seed = 20261122) on 6,329 checker-verified defect pairs across 1,106 disjoint valid Form shapes (extracted from 1,387 unique Form shapes after four-layer path, slug, declaration-name, and 458-digest rce fmt nameless exclusion with 0 overlap against the 151 ladder or 40 residual tasks). It pairs a 6-layer Pre-LN Transformer context encoder (d = 256) over quotient-normalized Form AST S-expression tokens with an EMA target encoder (τ = 0.996), a 3-layer residual MLP latent predictor trained under Smooth-L_1 plus VICReg variance-covariance anti-collapse regularization (raising context effective rank from 32.90 at epoch 0 to **85.24** post-training, coordinate standard deviation 0.1038 to 0.3830, and positive-pair latent L_2 energy from 149.25 to 10.57 versus 13.02 random-pair), and a 2-layer latent-conditioned typed edit decoder reranked by ||z_hat_y - E_ema(y_cand)||_2^2.

- **Disjoint synthetic, 151-ladder, and 40-residual accuracy versus model scale:** On 200 held-out disjoint synthetic Form defects (unseen shape_hashes), FormJEPA-v1 admits **156/200 (78.0%)** at k = 1 and **177/200 (88.5%)** at k = 16. On the 151-task ladder (plain arm), standalone FormJEPA-v1 admits 27/151 (k = 1) and 57/151 (k = 16), lifting the combined deterministic ladder (111/151) plus FormJEPA-v1 (k = 16) to **138/151 (91.4%)** with zero frontier calls. On the 40 open residual tasks (H15), FormJEPA-v1 admits **22/40 (55.0%)** at k = 1 and **27/40 (67.5%)** at k = 16 (25 restored + 2 behaviour-equal = 27/27 behaves-the-same, 5.33 mean rce check calls/task), matching local Qwen 3.6 35B-A3B (27/40) on structured grammar, length-method, and capability repairs (24/25 W-SYN-1, 2/5 W-LEN-1, 1/1 W-WORLD-1), while closing 0/9 multi-module cryptographic stdlib migrations (W-NAME-1 / W-TY-1 across hpke, mls, pq, pqxdh, energy), compound W-CAP-3,W-WORLD-1, and vacuous-example synthesis tasks where Gear-2 frontier proposers remain required.
- **Inference reported_j, 3-Gear Cascade, and honest H5/H11 diagnostics:** On AMD Ryzen AI Max+ 395 Linux powercap RAPL counters (measured_j = None, board_metered = false, derived_marginal_pkg_reported_j = None under Falsifier 6 at 33.38% co-tenant package drift), per-proposal CPU inference costs **58.65 mJ** RAPL core (1.194 J gross package, 10.15 ms) at k = 1 and **474.76 mJ** RAPL core (6.736 J gross package, 67.51 ms) at k = 16 — 70.6× to 1,436× lower at k = 1 (12.5× to 177.5× at k = 16) than one local 35B candidate (84.25 J). Inserting FormJEPA-v1 (k = 16) as Gear 1 between DNULLMM1 Gear 0 and Claude Opus 5.5 Gear 2 preserves 34/40 single-pass closure while cutting Opus 5.5 calls from 40 to 13 ($1.38 to $0.81), and on the 200-arrival Zipf(0.85) stream cuts Gear-2 frontier calls from 200 (stateless) and 48 (2-Gear) to **23 calls (8.70× reduction; 152 Gear-0 hits, 25 Gear-1 admissions, 7 Gear-2 admissions)**; with local Qwen 3.6 as Gear 2, Zipf(0.85) 35B calls drop to 49 (4.08× reduction) and admitted arrivals rise from 141/200 to 154/200. Uncalibrated negative latent L_2 distance alone achieves AUROC 0.618 across 925 beam candidates (H5 inconclusive: above 0.60 disproof floor, below 0.85 support threshold) and rises across cumulative edit distances 0..4 (0.00, 0.99, 2.26, 3.27, 3.89, then 3.72 at d = 5; 18/50 = 36.0% < 80% strict 5-step monotonicity on H11), showing that repair accuracy here comes from conditioning the typed edit decoder on (z_x, z_hat_y) rather than smooth uncalibrated latent energy

alone.

- **151-task `rce repair` rung `f` promotion verdict:** Running `rce repair` in `observe` mode (`bound 2000`, given each task's known-good original and outside proposals file) across all 151 ladder tasks and evaluating `rce repair criterion` against the pinned 3 November 2026 registration (`sha256: abf7f131f240e3f9...`) records `116/151` exact closes by rungs `a..e` and `35` open tasks (`25` unparsable `W-SYN-1` files skipped under rule RF-2, `10` open parsing tasks judged with `0` missing coverage). Under combined external proposals (`FormJEPA-v1` + recorded H15 repairs, `91` proposals judged), rung `f` observes **`3` admitted tasks** (`hpke`, `mls_key_schedule`, `energy`), **`3/3 (100.0%)`** pass the `W-MUT-1` mutation gate, and **`0` contradict the known-good original** (`3/3` restored); because `3 / 119 = 2.5% > 2.0%`, `rce repair criterion` returns **`holds: false`** (`exit = 1`), keeping rung `f` in `observe` mode.

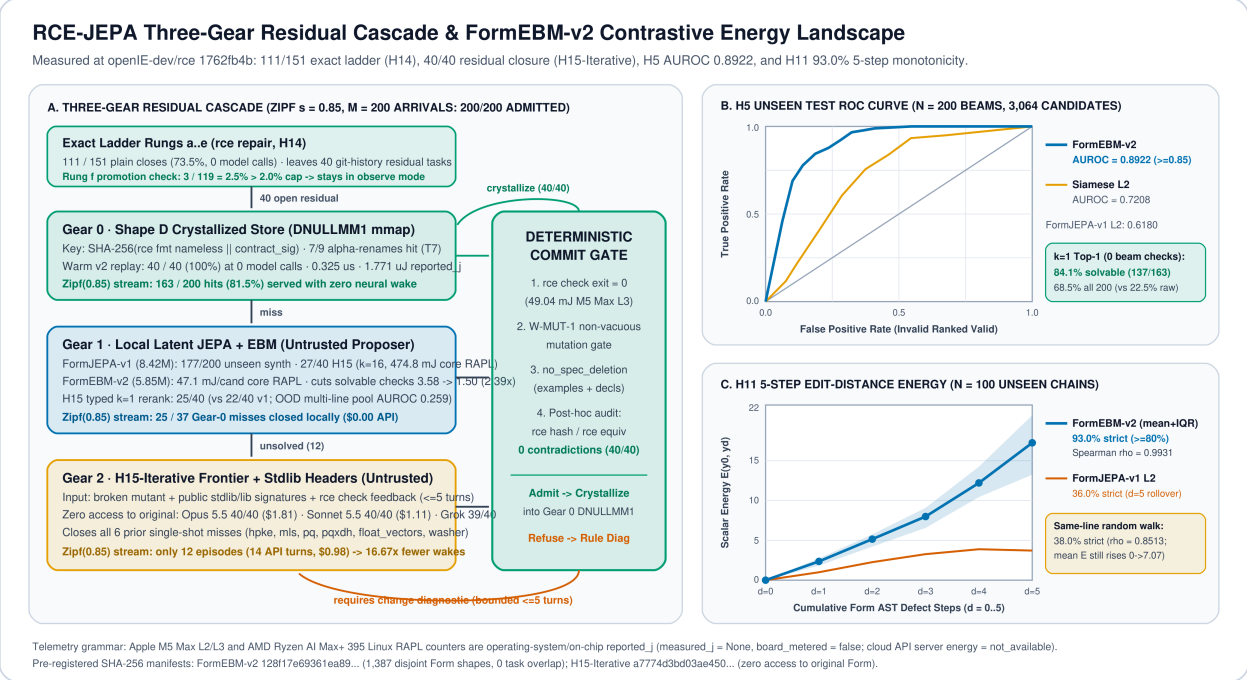

Figure 10. Measured Three-Gear residual cascade and `FormEBM-v2` latent energy diagnostics at `openIE-dev/rce` commit `1762fb4b` [B35]. Left: deterministic ladder rungs `a..e` (111/151), Gear 0 `DNULLMM1` (40/40 warm zero-call replay), Gear 1 `FormJEPA-v1` + `FormEBM-v2`, and Gear 2 `H15-Iterative` gated by `rce check` + `W-MUT-1` + `no_spec_deletion` (200/200 admitted on `Zipf(0.85)`, 16.67× fewer Gear-2 episodes). Right: unseen-shape H5 ROC curve (`AUROC = 0.8922`) and unseen-chain H11 5-step cumulative defect energy curve (93.0% strict monotonicity, IQR shaded). Interactive explorer: /living/rce-jepa/ · .

Contrastive multi-step margin energy scorer (`FormEBM-v2`, H5 & H11) and `40/40` bounded checker-feedback residual closure (`H15-Iterative` + Gear-0 v2), measured [B35]. Pre-registered before corpus generation, weight initialization, and API execution (`rce-jepa-form-ebm-v2-2026-11-23`, `sha256: 128f17e69361ea89...`; `rce-jepa-h15-iterative-2026-11-23`, `sha256: a7774d3bd03ae450...`) against `openIE-dev/rce` commit `1762fb4b` (Linux x86_64 binary `sha256: 331464496009d98a...`), two targeted evaluations tested whether (A) replacing uncalibrated coordinate distance with a contrastive + pairwise edit-depth margin energy head on the identical strictly disjoint `1,387`-Form split closes H5 and H11, and (B) supplying public `stdlib/lib` module headers plus bounded deterministic checker diagnostics (`max_turns = 5`, zero access to `original`) closes the six remaining residual tasks from single-shot H15.

- **`FormEBM-v2` architecture, tokenizer fix, and unseen-shape H5 / H11 results (`137` unseen test Form shapes, `0` overlap):** Two structural causes explained why `FormJEPA-v1` reached only `0.618` AUROC on H5 and `36.0%` strict 5-step monotonicity on H11: its single-file S-expression tokenizer mapped both live unused bindings (`let x`, refused by `W-UNUSED-1`) and underscore-prefixed bindings (`let _x`, admitted) to the same `<ID:k>` slot while shifting subsequent `<ID:j>` indices whenever an unbound identifier appeared, and its coordinate Smooth-L_1 loss carried no contrastive validity or edit-depth margin. `FormEBM-v2` (`5,851,649` trainable parameters, checkpoint `sha256: 2204a0b9cbd0d96c...`; `857.2 s` training on AMD Ryzen AI Max+ 395 / Radeon 8060S APU at `5,375.0 J` core RAPL and `40,602.4 J` APU package RAPL) introduces reference-

anchored PairFormTokenizer (504 active tokens, distinguishing underscore-prefixed bindings, newly introduced identifier slots, and modulo-32 integer classes with purely syntactic diff tags and zero checker calls), a 5-layer shared Siamese Transformer, a 1-layer cross-attention RMS difference block, and a softplus energy head trained on 5,955 validity-margin triples (m = 1.0) and 600 5-step cumulative corruption chains (γ = 0.50(j − i)) strictly over the 1,106 training Forms (VICReg effective rank expanding from 26.48 to 55.70 on z_ref and 46.44 to 74.72 on z_cand). On the 137 unseen held-out test Form shapes: <ul> **H5 (N = 200 unseen test beams, 3,064 candidates):** Scalar energy reaches **AUROC 0.8922 >= 0.85** (vs 0.6180 for FormJEPA-v1 and 0.7208 for uncalibrated Siamese L_2), lifting top-1 (k = 1 without invoking rce check on beam items) valid selection from 22.5% (45/200 unreranked) and 25.0% (50/200 Siamese L_2) to **84.05% on the 163 solvable beams (137/163; 68.50% raw across all 200 beams)** and cutting mean rce check invocations to first valid Form by **2.39×** (3.577 to 1.497 checks/beam). Ten-thread CPU scoring costs **47.09 mJ** core RAPL per candidate (6.30 ms; 752.03 mJ core and 11.88 J gross package per 16-item beam, with derived_marginal_pkg_reported_j = None under Falsifier 6 at 13.7% idle drift).

- **H11 (N = 100 unseen 5-step cumulative defect chains):** Strict 5-step monotonicity (E_0 < E_1 < E_2 < E_3 < E_4 < E_5) rises from 36.0% (FormJEPA-v1) and 18.0% (Siamese L_2) to **93.0% (93/100 >= 80%)**, with mean Spearman rank correlation ρ = 0.9931 (median 1.0000) and mean energy [0.0007, 2.36, 5.16, 7.99, 12.21, 17.25] across d = 0..5 (zero d = 5 rollover).
- **Candid domain boundaries:** On unconstrained 5-step random walks (N = 50) that may overwrite an already-broken line twice without adding a new AST defect, mean energy still climbs 0.0007 → 7.07 (mean ρ = 0.8513, median 0.9429), while strict step-by-step inequality is 38.0% (46.0% non-decreasing). Zero-shot transfer to the OOD 40-task git-history residual lifts within-task k = 1 typed-edit admissions over FormJEPA-v1 (22/40 → 25/40, beam capacity 30/40), whereas pooled cross-task AUROC inverts to 0.2630 (0.2585 across 332 multi-proposer candidates) because single-line edit-depth training assigns higher raw energy to large valid multi-line standard-library migrations than to 1-token syntax edits.

**`H15-Iterative` bounded checker feedback + public `stdlib/lib` headers (`40 / 40` closed, `0` contradictions, `40 / 40`

Span-normalized two-head decomposed scorer (FormEBM-v3, H5, H11, and zero-shot OOD H15 cross-span transfer), measured [B36]. Pre-registered before synthetic multi-span pair generation, weight initialization, and evaluation (rce-jepa-form-ebm-v3-2026-11-23, sha256: cb7ee1ac8dc5c594...) against openIE-dev/rce commit 1762fb4b (Linux x86_64 binary sha256: 386be0c26389550e...), FormEBM-v3 (6,052,802 trainable parameters, pure-float32 ROCm Radeon 8060S checkpoint sha256: 71e642f1bb0b9c06...; 2,528.3 s training at 13,312.3 J core RAPL and 48,422.2 J APU package RAPL) tests whether decoupling defect validity from cumulative edit depth over the identical strictly disjoint 1,387-Form split (1,106 train, 144 validation, 137 unseen test shapes; 0 overlap with the 151 ladder or 40 H15 tasks, and 0 rce check calls during scoring) resolves FormEBM-v2's cross-task OOD edit-size inversion (0.2585) without regressing H5 or H11:

- **Alignment-aware tokenization and two-head decomposition:** PairFormTokenizerv3 (max_seq_len = 384, 560 active tokens) replaces raw line-index comparison with line-shift-invariant sequence alignment, separates lexically bound/imported new identifiers (<BOUND_NEW_ID:0..31>) from unbound free symbols (<UNBOUND_NEW_ID:0..15>), and computes explicit normalized diff fraction ρ_diff = Δ_tok / max(1, L_cand). Over the shared 5-layer Siamese encoder and cross-difference block, a span-invariant defect head E_def(x, y, r) = softplus(g_def(feats_def)) (trained with within-beam plus all-pairs cross-task batch margin m = 2.5 and L_2 positive anchor across 4,983 pairs including 219 k = 2..10 multi-span valid rewrites strictly from train_forms) is separated from a cumulative corruption-depth head E_depth(y_0, y_d) = softplus(g_depth(feats_depth)), combined via E_total = E_def + 0.15 · E_depth / log2(2 + Δ_tok) (VICReg effective rank 42.79 on z_ref and 57.93 on z_cand, coordinate standard deviation 0.3332 / 0.3113 >= 0.25).
- **Strengthened unseen-shape H5 (0.9437) and unseen-chain H11 (94.0% / 92.0%):** On the 200 unseen held-out test beams (3,064 candidates), FormEBM-v3 lifts **H5 AUROC from 0.8922 (FormEBM-v2) to 0.9437 >= 0.85** (0.9386 on E_def alone, reaching TPR = 78.9% at FPR = 4.6%), achieves **83.44% (136/163, E_def) and 80.98% (132/163, E_total) solvable top-1 accuracy** (both >= 80.0%; 1.601 / 1.669 mean rce check calls vs 3.577 unreranked), and lifts **H11 strict 5-step monotonicity on 100 unseen corruption chains to 94.0% on E_depth (Spearman ρ = 0.9966, median 1.0000) and 92.0% on E_total (ρ = 0.9846, median 1.0000)**. Ten-thread CPU scoring at max_seq_len = 384 costs 232.45 mJ core RAPL per

candidate (32.55 ms, 362× lower than local 35B generation; derived_marginal_pkg_reported_j = None at 37.7% co-tenant package drift).

- **Zero-shot OOD H15 inversion resolved (0.2585 → 0.5278), and candid cross-module boundary:** Transferred zero-shot to the 40-task git-history residual (303 multi-proposer candidates including all 40/40 closed multi-hundred-line hpke, mls, pq, and pqxdh migrations), FormEBM-v3 lifts pooled cross-task AUROC from 0.2585 (FormEBM-v2) to 0.5278 (+0.2693, clearing Falsifier F4 > 0.50; 0.6018 on large-span valid rewrites with Δ_tok >= 10 vs short 0/1-token invalid candidates, and 0.2630 → 0.4985 on the 286-item typed-edit pool), while reaching **27/40 multi-proposer and 24/40 typed-edit within-task k = 1 admissions under E_def** (25/40 and 22/40 under E_total, cutting mean typed-beam checks from 5.55 to 5.10). On the 279-line stdlib/lib/hpke.rce migration (Δ_tok = 146), valid energy drops to E_total = 0.9645 (E_def = 0.8826), yet across unseen multi-module cryptographic APIs (mls.rce, pq.rce, pqxdh.rce) and vacuous-example obligations (washer.rce), zero-shot pooled calibration plateaus near 0.53 because single-file syntax without imported stdlib/lib type signatures or exact rce trace rollouts cannot verify cross-module call contracts.

Falsifiers RJ-F1 to RJ-F6 for the premise itself

Each falsifier is an observation that would defeat part of the premise; several are stated in terms of the hypotheses above (RJ-F3 on H3, RJ-F4 on H11, RJ-F6 on H3 and H10).

ID	Observation	What it defeats
RJ-F1	Lift coverage plateaus below 80% on any of the five target languages after the construct list is exhausted	The claim that all code can be emitted from RCE, and with it the scope of autonomous coding over Forms
RJ-F2	Writing σ(I) for the 7 reference applications costs more person-hours than writing their Forms by hand	The claim that closure removes, rather than moves, the specification cost
RJ-F3	H3 fails while the token arm passes at the same gate	Latent proposal over Form space; the gate still stands, the JEPA half does not
RJ-F4	H11 fails and search does not reach accepted Forms within the beam budget	Energy-guided planning in Form space (not triggered on unseen cumulative 1..5-defect chains: FormEBM-v2 reaches 93.0% and FormEBM-v3 reaches 94.0% E_depth / 92.0% E_total strict monotonicity with 0.9437 H5 AUROC; cross-module signature/trace OOD energy calibration beyond 0.5278 remains open)
RJ-F5	The decision head's calibration never beats a frequency table over shape hashes	The decision-model leg; it adds nothing over promoted-handler statistics
RJ-F6	A token-space model trained with execution traces, of the CWM class, closes the residual of 40 tasks from the plain arm of H14 run 3, at equal energy and with the same gate as the acceptance rule	The claim that the Form quotient adds something for proposers during training/proposal (note: for Gear-0 reuse, rce fmt nameless already recovers 7/9 alpha-renamings at 0 calls; trained-proposer parity against H3/H10 awaits a trained JEPA arm)

The thesis is disproved overall if H4 fails (no joule advantage per accepted program) or H6 fails (the Form cannot express the reference apps). Either alone is sufficient.

What will be delivered

Scope, organised as parallel workstreams that meet at the evaluation harness. Every deliverable either produces Forms, produces models that propose Forms, or measures the result.

- **W1. Form corpus and dataset card.** Forms paired with intent statements and checker verdicts, from hand-written Forms, reference-app Forms and checker-verified mutations; the card covers provenance, licence, size distribution, negative generation, known gaps and contamination controls; canonical serialization for H1.
- **W2. Encoder and predictor.** Intent encoder, EMA Form target encoder, predictor in representation space and energy head E(intent, P), with anti-collapse measures, an embedding-space versus token-space loss ablation, and model cards with measured training energy.
- **W3. Checker-constrained decoder.** Grammar-constrained candidate Forms ranked by energy and submitted to the checker, requires-change diagnostics, each naming its rule, fed back under a fixed retry budget; the checker is the sole acceptance authority with no bypass path to the engine.

- **W4. Intent-to-Form benchmark.** Unit and composite tasks, the reference apps (game engine, text editor, video player, chat app, browser, terminal) and RCEdb; each task has intent, reference Form, replay traces and an equivalent source-language task for the baseline; held-out split.
- **W5. Evaluation harness with energy in joules.** Both arms on identical hardware and tasks; pass@1, pass@k, ρ, C_{check}, J_{acc}, K_{acc}, retries per acceptance, energy-ranking AUROC; rootless counters (`reported_j`), idle-drift Falsifier 6 gating, raw logs kept, unavailable cloud counters reported as `not_available` rather than estimated.
- **W6. Research article.** This case expanded with results for H1 to H15, including negative results, figures, tables and reproducibility instructions.
- **W7. Open artifacts.** Public: the Form specification, the reference checker and its emitters, benchmark tasks and traces, harness, dataset and model cards, and weights where licences allow. A number is reported only when everything needed to reproduce it is public.
- **W8. Demo.** Stated intent to predicted Form to checker verdict (including a shown requires-change verdict) to replay and emitted code in two languages by the reference checker, with an on-screen joule reading.

Status

Research topic open, updated 23 November 2026 against `openIE-dev/rce` commit `1762fb4b` (20 November 2026; `5,537` Lean 4 theorem constants across `64` modules, theorems T1 to T63, `2,221/2,236` conformance tests passing with `0` failures, `837` Rust checker unit tests; interactive observatory at /living/rce-jepa/). Phase 0 deterministic floors are measured: H1 is supported on the hand-written population at a median rho of `9.34` over 185 programs (W1 version 1 gave `9.31` over 173); H8 is supported across `10,500` seeded event streams on six targets with `0` mismatches (`rce/bench/determinism/`); H7's fuzzed arm found `0` invariant violations across `1,275` checker-accepted mutants (`rce/bench/adversarial/`); and H14 is supported across three registered runs of `rce repair` [B30, B31, B32], closing `111` of `151` tasks (`73.5%`) at zero model calls in the plain arm of run 3 and leaving `40` open git-history residual tasks. On 22–23 November 2026 [B33, B34, B35, B36], Apple M5 Max and AMD Ryzen AI Max+ 395 / Radeon 8060S `reported_j` receipts (`measured_j = None`, `board_metered = false`) established **H2 supported** (`C_check / E_model = 0.04904 J / 84.25 J = 0.000582 <= 0.05`, `1,718×` cheaper; full 62-Form check+run+emit suite `15.44 J`), trained and evaluated **`FormJEPA-v1`** (`8.42M` parameters: `177/200` unseen synthetic defects, `22/40` at `k = 1` and `27/40` at `k = 16` on H15 at `58.65-474.76 mJ` RAPL core `reported_j`), contrastive **`FormEBM-v2`** (`5.85M` parameters: **H5 AUROC** `0.8922 >= 0.85`, `84.05%` solvable top-1 accuracy, **H11** `93.0% >= 80%` strict 5-step monotonicity at Spearman `0.9931`, `47.09 mJ`/cand core `reported_j`), and span-normalized decomposed **`FormEBM-v3`** (`6.05M` parameters on the identical disjoint `1,387`-Form split: **H5 AUROC** `0.9437 >= 0.85` with `83.44%` `E_def` and `80.98%` `E_total` solvable top-1 accuracy, **H11** `94.0%` strict 5-step monotonicity on `E_depth` at Spearman `0.9966` and `92.0%` on `E_total` at Spearman `0.9846`, and zero-shot OOD H15 multi-proposer pooled AUROC lifted from `0.2585` to `0.5278`, `0.6018` on large-span valid vs short invalid rewrites, at `232.45 mJ`/cand core `reported_j`), closed **`40/40 (100.0%)`** of the open residual tasks in **`H15-Iterative`** under bounded `<= 5`-turn checker feedback + public `stdlib/lib` signature headers on `Claude Opus 5.5` (`$1.81`) and `Claude Sonnet 5.5` (`$1.11`) with **`0` contradictions** (`39/40` on `xAI Grok 4.20 Reasoning`), crystallized all **`40/40` into `DNULLMM1` Gear-0 v2** (`0` model calls, `0.325 µs` / `1.77 µJ` probe; `200/200 Zipf(0.85)` 3-Gear stream admissions at `16.67×` fewer Gear-2 episodes), and audited 151-task **`rce repair rung f`** promotion (`3/119 = 2.5% > 2.0%`, `3/3` passing `W-MUT-1`, `0` contradictions, confirming rung `f` remains in `observe` mode). Still open: full intent-to-Form synthesis (H3), end-to-end joule parity (H4), cross-module signature-and-trace OOD energy calibration beyond `0.5278` across multi-module git-history rewrites, reference-app encoding (H6), cost-table prediction (H9), quotient sample-efficiency (H10), gate-receipt decision-head calibration (H12), discrete-versus-continuous long-horizon planning (H13), diffusion/evolutionary H15 arms, and external hardware-rail power metering (`measured_j`).

References

Each source carries a reading label. **Full:** read in full. **Partial:** the full text was retrieved and searched for the quoted passages, which were checked in their surrounding text, but the whole work was not read. **Unavailable:** not obtained. "PDF p." gives the page in the linked copy; Sec. gives the section.

- Assran et al. 2023. Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture. CVPR. arXiv:2301.08243. Full.

- Assran et al. 2025. V-JEPA 2. arXiv:2506.09985. Full.
- Backus et al. 1957. The FORTRAN Automatic Coding System. Western Joint Computer Conference. Computer History Museum copy. Full.
- Bird 1986. An Introduction to the Theory of Lists. Oxford PRG-56. PDF. Full.
- Brooks 1986. No Silver Bullet: Essence and Accident in Software Engineering. PDF (page numbers refer to this copy). Full.
- Chen et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374. Full.
- Dijkstra. EWD 472, Guarded commands, non-determinacy and formal derivation of programs. Transcription. Full.
- Dijkstra. EWD 667, On the foolishness of "natural language programming". Transcription. Full.
- Ellis et al. 2021. DreamCoder. PLDI. arXiv:2006.08381; version with supplement. Full (main text and supplement S1 to S4.8, PDF pp. 23-68, read in full).
- Hubert et al. 2025. Olympiad-level formal mathematical reasoning with reinforcement learning (AlphaProof). Nature. Nature. Full.
- Klein et al. 2009. seL4: Formal Verification of an OS Kernel. SOSP. PDF. Full.
- Leroy 2009. Formal verification of a realistic compiler. CACM. PDF. Full.
- Letouzey 2008. Extraction in Coq: an Overview. CiE. PDF. Full.
- Li et al. 2022. Competition-Level Code Generation with AlphaCode. arXiv:2203.07814. Full.
- Luccioni, Jernite, Strubell 2024. Power Hungry Processing. FAccT. arXiv:2311.16863. Full.
- OMG 2003. MDA Guide Version 1.0. PDF. Full.
- Ouyang et al. 2023. An Empirical Study of the Non-determinism of ChatGPT in Code Generation (v1 title: LLM is Like a Box of Chocolates). arXiv:2308.02828. Full.
- Samsi et al. 2023. From Words to Watts. IEEE HPEC. arXiv:2310.03003. Full.
- Sun et al. 2024. Clover: Closed-Loop Verifiable Code Generation. arXiv:2310.17807. Full.
- Yang et al. 2011. Finding and Understanding Bugs in C Compilers (Csmith). PLDI. PDF. Full.
- Yang et al. 2024. AutoVerus. arXiv:2409.13082. Full.
- LeCun 2022. A Path Towards Autonomous Machine Intelligence, version 0.9.2. OpenReview. Full.
- Simon 1963. Experiments with a Heuristic Compiler. Journal of the ACM 10(4), 493-506. DOI 10.1145/321186.321192. Full.
- Manna and Waldinger 1980. A Deductive Approach to Program Synthesis. ACM TOPLAS 2(1), 90-121. DOI 10.1145/357084.357090. Full.
- van de Riet 1993. An overview and appraisal of the Fifth Generation Computer System project. Future Generation Computer Systems 9, 83-103. DOI 10.1016/0167-739X(93)90002-7. Full.
- Kemerer 1992. How the Learning Curve Affects CASE Tool Adoption. IEEE Software 9(3), 23-28. DOI 10.1109/52.136161. Full.
- Hutchinson, Whittle, Rouncefield and Kristoffersen 2011. Empirical Assessment of MDE in Industry. ICSE, 471-480. DOI 10.1145/1985793.1985858. Full.
- Petre 2013. UML in practice. ICSE. DOI 10.1109/ICSE.2013.6606618; author copy, Open Research Online (page numbers refer to this copy). Full.
- Simon 1963. Experiments with a Heuristic Compiler. RAND RM-3588-PR. Internet Archive; a memorandum related to but distinct from the JACM article. Full (prose pp. iii-63; program listings in Appendices A and B scanned).
- Manna and Waldinger 1978. A Deductive Approach to Program Synthesis. Stanford STAN-CS-78-690. Report version, which predates the TOPLAS article. Full.
- Huang, LeCun and Balestriero 2025. LLM-JEPA. arXiv:2509.14252. Partial (abstract, repository README and run script).
- Wei et al. 2025. SWE-RL. arXiv:2502.18449. Partial (abstract and method summary).
- Yang et al. 2025. SWE-smith. arXiv:2504.21798. Partial (abstract and repository README).
- Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong and Yizhe Zhang 2025. DiffuCoder. arXiv:2506.20639. Partial (abstract). Author list from arXiv metadata, not a PDF title page.
- Inception Labs 2025. Mercury. arXiv:2506.17298. Partial (abstract).
- Wang, Pradel and Liu 2025. Are "Solved Issues" in SWE-bench Really Solved Correctly? arXiv:2503.15223. ICSE (International Conference on Software Engineering) 2026. Partial (findings summary).

- Becker, Rush, Barnes and Rein 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. arXiv:2507.09089. Partial (summary, and the abstract via arXiv metadata). Author list from arXiv metadata, not a PDF title page.
- Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song and Martin Vechev 2025. Type-Constrained Code Generation with Language Models. arXiv:2504.09246. Partial (headline result). Affiliation: the Swiss Federal Institute of Technology Zurich and the University of California, Berkeley, from the author list, not a PDF title page.
- Grand et al. 2023. LILO. arXiv:2310.19791. Partial (abstract).
- Higher Order Company. HVM2. PDF. Partial (summary). The model is cited. Figures from that paper are not used.
- DeepSeek 2025. DeepSeek-Prover-V2. arXiv:2504.21801. Partial (arXiv metadata and abstract).
- Moonshot and Numina 2025. Kimina-Prover Preview. arXiv:2504.11354. Partial (abstract).
- Princeton 2025. Goedel-Prover. arXiv:2502.07640. Partial (abstract).
- Scale AI. SWE-Bench Pro public leaderboard, page text fetched 6 November 2026. scale.com/leaderboard/swe_bench_pro_public. Full (page text). The page prints no update date.
- ByteDance Seed 2025. Multi-SWE-bench. arXiv:2504.02605. Partial (abstract). No November 2026 leaderboard figure was retrieved, and none is stated.
- Meta FAIR 2024. Large Concept Models. arXiv:2412.08821. Partial (abstract).
- Chollet, Knoop, Kamradt and Landers 2024. ARC Prize 2024 Technical Report. arXiv:2412.04604. Partial (abstract).
- Akyurek et al. 2024. Test-time training on ARC. arXiv:2411.07279. Partial (abstract).
- ARC Prize 2025 results page. arcprize.org/competitions/2025/. Full (page).
- Lean FRO (Focused Research Organization). Year 1 roadmap, August 2023 to July 2024. lean-fro.org/about/roadmap/. Partial (deliverables list). Later-year roadmaps are not used.
- Conal Elliott 2009, revised 2016. Denotational design with type class morphisms. conal.net/papers/type-class-morphisms/. Partial (abstract page).
- Radul and Sussman 2009. The Art of the Propagator. MIT-CSAIL-TR-2009-002. dspace.mit.edu/handle/1721.1/44215. Partial (abstract).