

Active Assembly Theory and Practice

Assembly-Code Observations for Active Inference Demonstrations

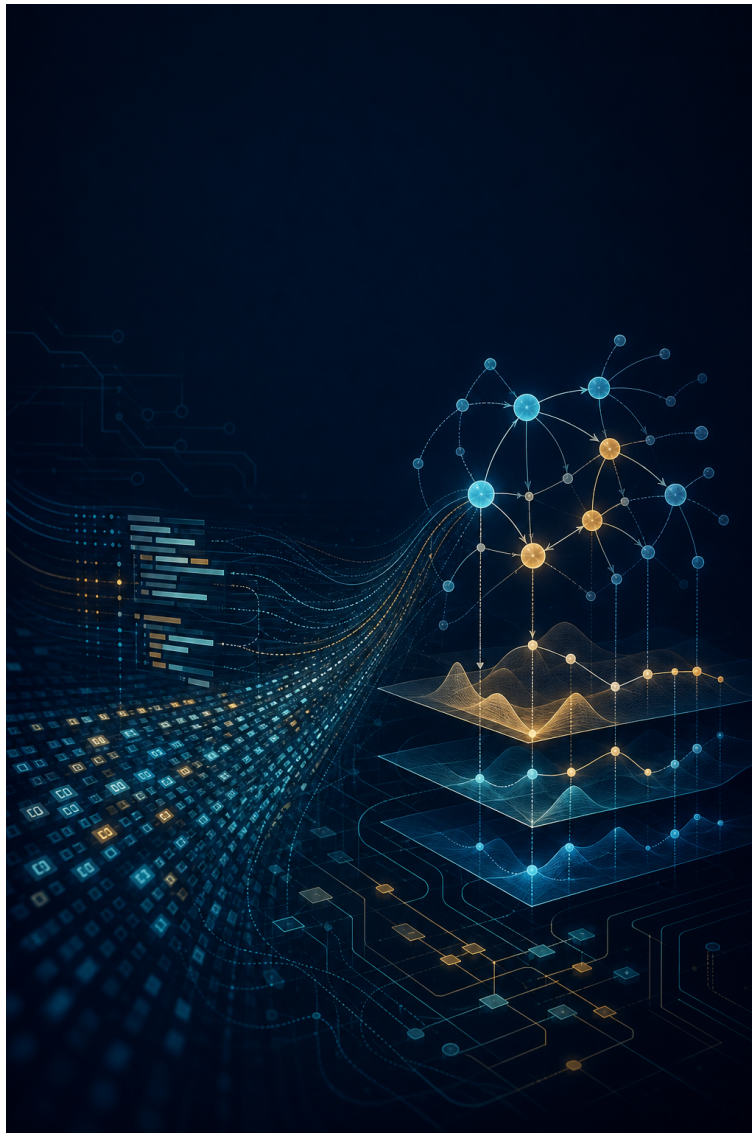
Daniel Ari Friedman

Active Inference Institute

`daniel@activeinference.institute`

ORCID: 0000-0001-6232-9096

2026-10-19



Contents

1	Abstract	3
2	Introduction	4
2.1	Evidence map and contribution boundary	4
2.1.1	Recorded research architecture	5
3	Methodology	7
3.1	Assembly evidence and notation	7
3.1.1	Executable model specifications	7
3.1.2	Instruction evidence and bounded state	8
3.1.3	Typed evidence channels and factorized scoring	10
3.1.4	Observation semantics and protected evidence	10
3.1.5	Fixture-design coverage	12
3.1.6	Bounded transfer functions and control-flow formalism	14
3.1.7	Mechanically checked closed-world semantics	16
3.1.8	Cross-architecture adapter falsification surface	18
3.1.9	Memory operands and dynamic access provenance	20
3.2	Temporal, active, and evaluated inference	21
3.2.1	Observation-only parameter learning	24
3.3	Formal description	25
3.3.1	Action-conditioned sensing	26
3.3.2	Exact finite-horizon information-per-cost analysis	33
3.3.3	Protocol-v2 artifact envelope and manuscript closure	36
3.4	Assembly state, normalized intermediate representation, and flags	36
3.5	Formal correspondence and transfer receipts	38
3.6	Quotient classes and co-primary classifier portfolio	40
3.6.1	Exact intervention power and finite separating sets	42
3.7	Measured sensing costs and finite policy certification	45
3.8	Frozen blind replication harness	46
4	Results	49
4.1	Prospective inference from executable GNN models	49
4.2	Trace and comparison diagnostics	49
4.3	Measured phase decoding	51
4.3.1	Leakage-safe held-out estimation	53
4.3.2	Observation-only ordered-HMM learning	53
4.3.3	Typed observation expansion	53
4.3.4	Statistical reporting contract	54
4.4	Where temporal dynamics matter	57
4.5	How well we segment phases	58
4.6	Which phases are recoverable	58
4.7	Robustness to the small corpus	61
4.8	Action-conditioned sensing	61
4.9	Open-roadmap evidence tranche	61
4.10	Theorem-or-countertheorem outcome	65
4.11	Formal counterexample search	67
4.12	Active-sensing stress study	68
5	Discussion	70
5.1	What the visual atlas adds to the scientific argument	71
6	Validation and Scholarship Boundary	73
6.1	Open-roadmap receipt gates	79
6.1.1	Model, execution and sensing receipts	79
6.1.2	External source and label contracts	79
6.1.3	Replay, formal status and acquisition gates	79

7	Limits, validity, and scientific roadmap	81
7.1	Long-horizon scientific program	83
7.2	Current promotion ledger for P3–P17	83
7.2.1	Benchmark, formal and replication evidence (P3–P7)	83
7.2.2	Mechanism, robustness and transfer evidence (P8–P12)	84
7.2.3	External evidence and promotion gates (P13–P17)	84
7.3	Generated figure atlas	85
8	References	87

1 Abstract

Assembly traces can support explicit probabilistic inference without being treated as recovered program meaning. **Active Assembly Theory and Practice** makes that distinction executable: transparent x86-64 fixtures are decoded with Capstone, traced under bounded Unicorn emulation, and mapped to typed observations that distinguish missing, unseen, and unsupported evidence. Two Generalized Notation Notation (GNN) specifications define passive-phase and prospective sensor models whose tensors drive real pymdp execution. A native categorical Julia/RxInfer companion consumes the passive model’s same parameters [Friedman, 2026, Heins et al., 2022, Bagaev et al., 2023]. The sensor Agent selects an action before fixture evidence is acquired. Independent finite calculations check beliefs and policy scores; native smoothing uses the same parameters and first-observation convention. Every inference comparator and fitted diagnostic also consumes an upstream-parsed numeric GNN specification, with source-bound runtime instances and rendered model graphs.

A corpus of 10 authored fixtures supports complete-fixture exclusion diagnostics. At the fixed informative likelihood, the memoryless decoder outperforms the temporal filter; temporal advantages appear under synthetic observation ambiguity. Designed category ambiguity remains a visible failure case. The separate Gaussian and categorical Bethe free-energy diagnostics are not compared numerically with policy expected free energy.

The quotient benchmark contains 64 transformation-family identifiers and 4096 rows, with 16 families held out from model fitting. The co-primary portfolio includes an interpretable classifier, a temporal probabilistic model, a fixed one-layer tanh recurrent classifier, and surface baselines; the temporal endpoint reaches balanced accuracy 0.775 and macro-F1 0.772 on this frozen bounded benchmark. Held-out identifiers share authored template lineages with the fitting data. Family-resampling intervals and conditional randomization diagnostics therefore describe this finite inventory; the retrospective protocol correction withholds superiority claims. A complementary pinned static-only corpus supplies 5 licensed source projects and 100 source-contract rows, including 40 lexical witnesses, 40 contract-directed near misses, and 20 identity controls. The corpus establishes provenance and coverage; it does not execute arbitrary external binaries or support application-effectiveness claims.

Selected normalized machine-state relations are checked by Python, native execution, and a bounded Lean fragment. Exact finite sensing certificates and source-bound reader artifacts make assumptions and failures inspectable. Independent physical replication remains unearned. The contribution is a reproducible substrate for testing assembly-to-inference mechanisms, with explicit limits on semantics, statistical generalization, and external validity.

2 Introduction

Assembly-analysis workflows sit between a machine-level instruction set and the higher-level hypotheses an analyst wants to test. A disassembler recovers a candidate sequence of variable-length instructions from bytes; an emulator can then expose a bounded dynamic path and register state; a downstream model may summarize those observations without pretending that the summary is a complete program meaning. The x86-64 architecture manuals provide the reference frame for instruction encodings and architectural effects [Intel Corporation, 2026]. Capstone provides the programmable static decode [Capstone Engine Project, 2026], Unicorn provides instruction-level emulation [Unicorn Engine Project, 2026], and larger systems such as angr can add symbolic or control-flow analyses when the research question requires them [angr Project, 2026, Shoshitaishvili et al., 2016].

This project deliberately studies the seam between those layers. Its canonical fixture experiment uses a Capstone–Unicorn core and keeps the byte program small enough that every byte, address, mnemonic, operand, category, and register result can be inspected. Optional adapter slots for Rizin, radare2, LLVM objdump, Keystone, and NASM are recorded as provenance rather than silently promoted to executed evidence [Rizin Project, 2026, radare2 Project, 2026, LLVM Project, 2026b, Keystone Engine Project, 2026, NASM Project, 2026]. The resulting question is narrow but testable: can an assembly trace be converted into an explicit categorical observation stream that supports reproducible variational and temporal inference diagnostics?

The contribution is methodological: it is not a general reverse-engineering algorithm or a new disassembler. The package provides (i) a source-bound static/dynamic trace contract and typed observation schema that preserve missing, unseen, static, and dynamic states; (ii) a width-aware normalized x86-64 machine state with explicit register, flag, memory, stack, control-flow, and unsupported states; (iii) a Python/Capstone/Unicorn/Lean correspondence surface for a declared user-mode integer subset, with theorem-backed and computational checks separated; (iv) a quotient benchmark with disjoint transformation-family identifiers and a co-primary classifier portfolio, calibration, clustered uncertainty, and leakage controls; (v) a pinned, permissively licensed, static-only source-contract track that records source, compiler, object, and disassembly provenance without executing external binaries; (vi) measured deterministic costs and exact finite active-sensing certificates; and (vii) a label-separated local replication harness whose independent-replay state cannot be fabricated by a local rerun. Real Python, Julia, and Lean execution paths, negative controls, manuscript variables, figures, captions, and validation receipts are regenerated from the same source-bound artifacts. This architecture allows stronger evidence to be added without silently changing the endpoint or claim boundary.

The experiment remains intentionally bounded. The canonical Active Inference fixture uses hand-authored phase states and fixed likelihoods; the temporal corpus grows to 10 transparent fixtures under a frozen 4/2/4 training/development/held_out family split but remains hand-labelled; the larger quotient benchmark is still synthetic, its family identifiers share authored template lineages across splits, and observations are dependent within traces; the external track provides source contracts and static coverage rather than behavioral labels; and the formal layer covers a declared subset rather than the complete architecture. These constraints are part of the result: the package makes the assumptions, failure states, and missing evidence inspectable instead of averaging them away.

2.1 Evidence map and contribution boundary

The paper should be read as a chain of evidence objects rather than as one undifferentiated claim. Static decoding establishes what the bytes can be parsed as; bounded execution records what a transparent fixture did under a declared intervention; the normalized reference path states which effects are modeled; Lean checks only the formal fragment named in its receipt; the statistical layers evaluate fixed endpoints on declared splits; and the document pipeline records how those results become a citable artifact. The architecture figure makes those boundaries visible so that a successful downstream render cannot be mistaken for a successful upstream scientific claim.

Evidence surface	What is established here	What remains outside the claim
Static and dynamic fixture path	Real Capstone decoding and bounded Unicorn instrumentation on transparent repository-owned inputs	Arbitrary-binary execution, complete path recovery, and general reverse engineering
Normalized state and Lean fragment	A width-aware, bounded operational contract and selected kernel-checked algebra	Complete x86-64 semantics, ABI behavior, exceptions, and unrestricted aliasing

Evidence surface	What is established here	What remains outside the claim
Quotient and model portfolio	Source-bound finite labels, disjoint family identifiers, calibration, and conditional clustered diagnostics	Independent source-lineage generalization, universal identifiability, and post-hoc model selection
External and replay surfaces	Provenance, static-only acquisition safeguards, and the recorded controlled replay outcome	Application effectiveness, independent physical replication, and independent reimplementations
Document and receipt pipeline	Reproducible source-to-render lineage with explicit unavailable and negative states	Peer review, adoption, and scientific validity inferred from engineering gates

2.1.1 Recorded research architecture

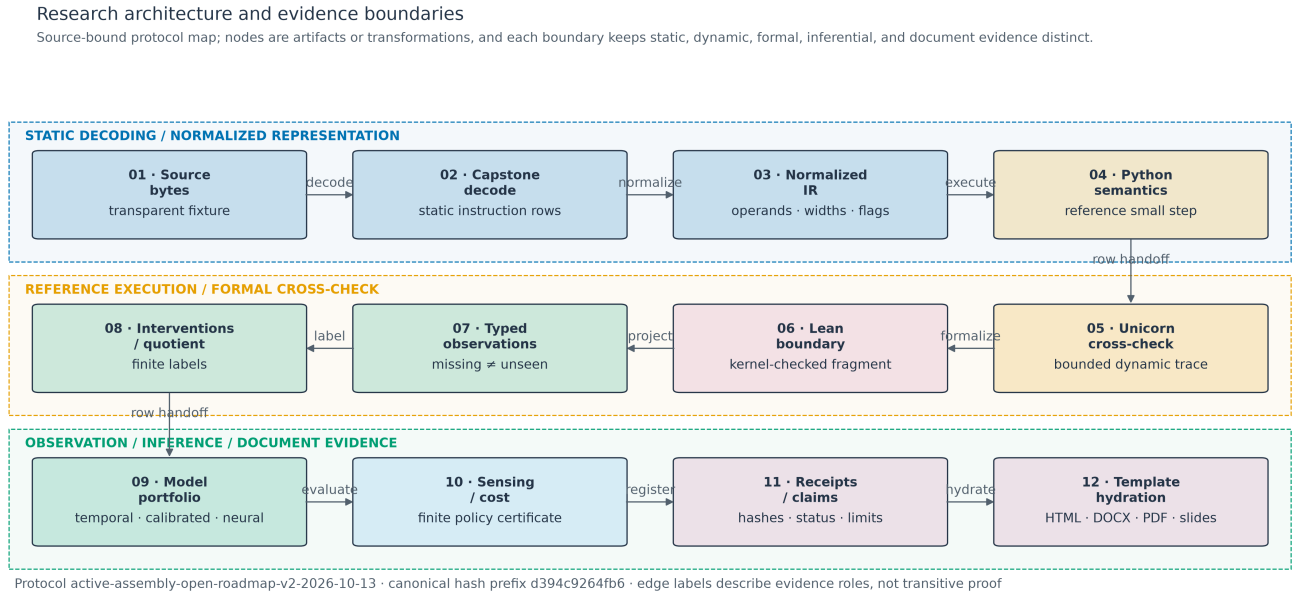


Figure 1: The research architecture maps the protocol active-assembly-open-roadmap-v2-2026-10-13 from source bytes through Capstone static decoding, normalized instruction representation, Python reference semantics, bounded Unicorn execution, the Lean formal boundary, typed observations, finite interventions and quotient labels, the co-primary model portfolio, sensing and cost evaluation, receipt/claim registration, and template hydration. Static, dynamic, formal, inferential, and document-production regions are outlined separately; the protocol hash prefix is d394c9264fb6. Edge annotations identify whether a relation is kernel-checked, computationally cross-checked, inferential, or provenance-preserving. The visual is an evidence map for this bounded technical monograph. It does not make the pipeline transitive proof, complete x86-64 semantics, universal identifiability, or application validity.

The figure is therefore both a reader’s map and a claim-control device. Its arrows describe the recorded relationship between artifacts, not a transitive proof that every representation preserves program meaning. The relevant unit of interpretation is always the receipt named by the adjacent text.

The broader package roadmap is intentionally larger than the implemented fixture. Binary-analysis platform work such as BAP and GTIRB motivates intermediate-representation and tool-interchange questions [Brumley et al., 2011, Schulte et al., 2019]. Full-suite and lifting/decompilation packages such as Ghidra, Qiling, Miasm, Dyninst, RetDec, and McSema are cited as future adapter surfaces, not as current evidence-producing backends [National Security Agency Research Directorate, 2026, Qiling Framework Project, 2026, CEA SEC, 2026, Dyninst Project, 2026, Avast, 2026, Trail of Bits, 2026].

The Active Inference bridge is framed as an observation problem. Machine-code events are not treated as mental states or semantic ground truth. Instead, each decoded instruction contributes a categorical observation under an

explicit likelihood matrix. Posterior beliefs over latent program-state classes can then be updated as the observation stream arrives, and the temporal extension can be tested against memoryless and identity-transition baselines.

The theoretical connection is deliberately narrow. The Free Energy Principle motivates variational objectives as a way to relate evidence, model structure, and belief updating [Friston, 2009]. Active inference provides a process-theoretic account of belief propagation under generative models [Friston et al., 2017]. Hidden Markov models supply the standard forward, Viterbi, and forward–backward machinery used for the temporal layer [Rabiner, 1989], while calibration and fixed-parameter evidence are reported with their own statistical caveats [Guo et al., 2017, Kass and Raftery, 1995]. This project borrows that computational grammar for transparent assembly traces; it does not claim that a byte fixture is a living agent or that reverse engineering has been automated.

3 Methodology

The demonstrator starts with a fixed x86-64 fixture whose instruction sequence is known and auditable. Capstone decodes the bytes into address, mnemonic, operand, and byte-span records. Unicorn then executes the same bytes under a bounded instruction count and records the executed addresses plus pre/post snapshots of all sixteen general-purpose registers, RIP, and EFLAGS. The Python reference state additionally carries definedness for modeled flags, bounded byte memory, bounded stack bounds, the decoded instruction, and an explicit execution status. The final register state is retained as a summary, not as the only semantic check. These evidence streams are joined into trace schema 1.4.0 with sample hash `bc3c4c15ae736188e6b91e7ff1a5a209367ada96cb31b245e5297e14fb80b44b`. The bundle records 3 available package adapters out of 16 total adapter slots, including unavailable optional packages as explicit provenance rather than silent assumptions; the package provenance is visualized in fig. 15.

The feature layer maps instruction mnemonics into 6 observation categories: load immediate, arithmetic update, bitwise transform, comparison, control transfer, and other. These observations are supplied to a discrete likelihood matrix over 5 latent program-state classes: setup, accumulator update, representation transform, guard check, and exit. The posterior update uses the real pymdp Agent’s state-inference interface, producing one belief vector and one comparison free-energy diagnostic per observation step. This identity-transition comparator retains a separate NumPy/softmax regression reference; it is not the prospective sensing model. The end-to-end observation pipeline is visualized in fig. 14, and the typed trace feature matrix in fig. 16.

3.1 Assembly evidence and notation

3.1.1 Executable model specifications

The canonical categorical specifications are `configs/gnn/passive_phase.md` and `configs/gnn/fixture_sensor.md`. The configured Generalized Notation Notation implementation [Friedman, 2026] parses their numeric tensors before a strict local compiler admits the supported finite profile. The compiler checks declared axes, distributions, dependencies and policy enumeration; the upstream revision, imported-source hashes and tensor digest accompany each study. The software citation identifies a documented source edition; the execution receipt identifies the parser revision actually used for the current run. A matching software version string alone does not establish source identity. A is observation likelihood, B is action-conditioned transition, C contains dimensionless utilities, D is the initial prior and E is the policy prior. The first observation conditions D before any transition. The passive categorical Python and native Julia paths consume the same parameter export; the prospective sensor model executes through pymdp.

The same source requirement extends to every inference comparator and fitted diagnostic. The generated catalog contains 14 model profiles and 2299 distinct numeric instances at `../output/gnn/inference/README.md`. Besides the two canonical specifications, these profiles declare the phase HMM, identity comparator, Gaussian event-code model, factorized and hierarchical likelihoods, finite sensor channels, retrospective ordering weights, logistic and recurrent classifiers, Markov classifier, Platt calibration, and linear cost calibration. A profile defines an architecture; a numeric instance binds its current parameters and training context. These counts describe software artifacts, not independent samples.

Each adapter consumes parameters returned by the configured upstream GNN parser. Its declared axes and probability constraints are checked before inference; fitted parameters are retained as specialized GNN sources. The upstream graph renderer and a larger-label rendering of the same parsed topology accompany every profile. Python and Julia adapters implement the declared bounded architectures. This extends source binding across the study without claiming universal GNN code generation or turning a discriminative classifier into an active-inference agent.

For this fixed-parameter profile, each policy’s negative expected free energy is the finite-horizon sum of expected dimensionless utility and modality-wise expected state information gain. With policy precision γ and policy prior E , the Agent uses

$$q(\pi) = \text{softmax}_{\pi}[\gamma(-G(\pi)) + \log E(\pi)].$$

Larger returned negative-EFE scores are favored; the reported EFE reverses their sign. Parameter learning and parameter information gain are disabled. This is the predictive policy objective exercised through pymdp, rather than the adaptive dynamic-programming optimum of the separate finite sensing experiment [Heins et al., 2022]. The executable software version and admitted model-language declaration are recorded separately in the upstream receipt.

In the bounded sensor POMDP, the real Agent infers states and policies and selects an action before the environment performs the selected static or dynamic fixture acquisition. Evaluation identity is recorded after decisions; missing acquisition and observed-but-unseen evidence are different outcomes. Independent finite references and A/B/C/D/E interventions test implementation behavior. These authored toy models do not establish learned semantics, general reverse engineering or external validity. Their policy EFE scale is distinct from both native Bethe free-energy diagnostics below.

3.1.2 Instruction evidence and bounded state

The byte fixture is an instruction-semantics test case, not an opaque binary sample. In x86-64, an instruction is a variable-length encoding whose bytes, decoded operands, architectural effects, and dynamic execution are related but not interchangeable. Intel’s architecture manuals provide the reference vocabulary for instruction encodings and register effects [Intel Corporation, 2026]. Capstone’s detail interface exposes instruction identifiers, operand kinds, implicit register reads/writes, and architecture-specific operand metadata [Capstone Engine Project, 2026, Quynh, 2014]. Unicorn complements that static view with a CPU-state transition interface and code/memory instrumentation hooks [Unicorn Engine Project, 2026, Quynh and Vu, 2015]. The implementation preserves those layers separately: Capstone supplies the static decode, Unicorn supplies the bounded address/register trace, and the feature mapper supplies the deliberately small observation vocabulary. This separation is what makes a later semantic or control-flow extension auditable rather than silently replacing decoded assembly with a guessed high-level label.

Let $b_{0:L-1}$ be the fixture byte string and $a_0 = 0x1000$ the load address. The static decoder is represented as

$$\delta_{CS}(b, a_0) = [(a_i, \ell_i, \text{mnemonic}_i, \text{operands}_i, \text{bytes}_i)]_{i=0}^{N-1}, \quad (1)$$

where a_i is an instruction address, ℓ_i its decoded byte length, and $N = 4$ for the selected fixture. The offsets are therefore recovered from the instruction lengths; they are not inferred from a fixed-width word assumption. The bounded dynamic record is

$$\tau = (r_0, r_1, \dots, r_T; a_0^{\text{exec}}, a_1^{\text{exec}}, \dots, a_{T-1}^{\text{exec}}), \quad (2)$$

where r_t is the captured register state and a_t^{exec} is the address observed by Unicorn’s code hook. The join is address-based: a static instruction is marked executed exactly when its address occurs in the bounded dynamic trace. That rule is intentionally weaker than claiming complete path or control-flow recovery.

lst. 1 is generated from these live Capstone and Unicorn records for architecture `x86_64`; the machine-readable source is `../output/data/assembly_listing.json`. The left fields preserve the assembly evidence; the trailing fields are the categorical projection consumed by the inference model.

Listing 1 Source-bound x86-64 fixture listing; static decode and bounded dynamic annotations are generated from the canonical trace bundle.

```

; fixture: x86_accumulator / architecture: x86_64
; Capstone: static bytes, mnemonics, register access
; Unicorn: executed addresses
; Model inputs: category and observation_id
0x1000 b8 01 00 00 00      mov eax, 1
; reads=- / writes=eax
; category=load_immediate / obs_id=0 / executed=yes
0x1005 83 c0 02          add eax, 2
; reads=eax / writes=rflags, eax
; category=arithmetic_update / obs_id=1 / executed=yes
0x1008 d1 e0            shl eax, 1
; reads=eax / writes=rflags, eax
; category=bitwise_transform / obs_id=2 / executed=yes
0x100a 83 e8 01          sub eax, 1
; reads=eax / writes=rflags, eax
; category=arithmetic_update / obs_id=1 / executed=yes

```

Assembly state plate: listing → state transition → bounded effects

Transparent fixture evidence; widths and flags come from the normalized formal step, dynamic registers from bounded execution, and memory rows from the bounded hook receipt.

Decoded listing and normalized instruction fields

address	bytes	opcode family	instruction	widths (op → state)	flag read / write	next RIP
0x1000	b8 01 00 00 00	load_immediate	mov eax, 1	32 → 64	— / —	0x1005
0x1005	83 c0 02	arithmetic_update	add eax, 2	32 → 64	— / CF PF AF ZF SF OF	0x1008
0x1008	d1 e0	bitwise_transform	shl eax, 1	32 → 64	— / CF PF AF ZF SF OF	0x100a
0x100a	83 e8 01	arithmetic_update	sub eax, 1	32 → 64	— / CF PF AF ZF SF OF	0x100d

Bounded register / PC transitions

step	RAX	RSP	RIP	EFLAGS
1	0x0→0x1	0x101000	0x1000→0x1005	0x2
2	0x1→0x3	0x101000	0x1005→0x1008	0x2→0x6
3	0x3→0x6	0x101000	0x1008→0x100a	0x6
4	0x6→0x5	0x101000	0x100a→0x100d	0x6

Bounded byte-memory effects

event	access	instruction	address	size	value
0	write	0x1000	0x101020	4	7
1	read	0x1008	0x101020	4	7

closed-world boundary

unsupported instructions, undefined flags, exceptions, and out-of-bounds memory return structured status

formal status: validated_bounded
fixture: x86_accumulator

RIP denotes the program counter. The panel is an exact fixture audit, not a complete x86-64 semantics claim.

Figure 2: The assembly-state plate shows 4 decoded instructions and 24 receipt step rows from the transparent x86_accumulator fixture. The listing panel reports addresses, bytes, decoded operands, operand widths, flag reads and writes, and next program counters. The state panel shows bounded register and RIP transitions; the memory panel contains 2 dynamic hook events. The boundary panel keeps unsupported instructions, undefined flags, exceptions, and out-of-bounds memory explicit. Values are exact fixture observations under the declared interventions, with no inferential confidence interval. The plate documents a normalized state contract and a transparent assembly listing; it does not establish complete x86-64 semantics, arbitrary alias analysis, ABI behavior, or execution safety for external binaries.

The state plate (fig. 2) is the visual companion to lst. 1. It is generated from the same listing and correspondence receipts, but it keeps the evidence types visually separate: bytes and decoded operands are static observations, register and program-counter snapshots are bounded dynamic observations, and flag/memory annotations are included only when the contract defines them. An unsupported or out-of-bounds transition is a status-bearing result, not a fabricated state. This layout is useful for reviewing the mechanics of a single step without suggesting that a short fixture is a complete architectural model.

The fixture exercises four distinct pieces of x86-64 notation: an immediate load into **EAX**, an arithmetic update, a shift-based representation transform, and a second arithmetic update. A write to a 32-bit register such as **EAX** also determines the corresponding 64-bit **RAX** state according to the x86-64 architectural rules [Intel Corporation, 2026]. Unicorn reports the resulting final register state **RAX** = 5; this is a concrete execution check on the bytes, not a claim that the mnemonic sequence constitutes a recovered program meaning. The feature map then applies

$$\begin{aligned} o_i &= \phi(\text{mnemonic}_i, \text{operands}_i) \in \mathcal{O}, \\ \mathcal{O} &= \{\text{load-immediate, arithmetic-update, bitwise-transform,} \\ &\quad \text{comparison, control-transfer, other}\}, \end{aligned} \tag{3}$$

so the model sees a declared categorical channel rather than raw opcode bytes. In particular, **add**, **sub**, and their related update mnemonics share the **arithmetic_update** channel, while **shl/shr**-family instructions share the **bitwise_transform** channel. This is a transparent pedagogical abstraction; it does not encode flags, memory aliasing, calling conventions, or complete instruction semantics.

3.1.3 Typed evidence channels and factorized scoring

The roadmap’s first expansion retains the discarded evidence as a typed record rather than silently changing the six-category benchmark. The schema version is 1.0.0 and contains 8 fields (7 static fields plus the dynamic execution status). The field families are **instruction family**, **operand class**, **register read set**, **register write set**, **flag effect**, **memory addressing mode**, **branch target class**, **dynamic status**. Instruction family and operand class are syntactic classifications over the decoded mnemonic and operand metadata; register read/write sets and flag effects are taken from Capstone’s detail API; memory addressing mode is a bounded classification of explicit base/index/displacement metadata; branch target class is internal, external, return, indirect, or none relative to the decoded span; and dynamic status is the address-membership join to the bounded Unicorn code trace [Intel Corporation, 2026, Capstone Engine Project, 2026, Unicorn Engine Project, 2026]. None of these fields is a phase label.

For instruction i , the typed record is

$$\mathbf{x}_i = (f_i, o_i, R_i, W_i, \varphi_i, m_i, c_i, d_i), \tag{4}$$

where f_i is instruction family, o_i operand class, R_i/W_i the canonicalized register read/write sets, φ_i flag effect, m_i memory addressing mode, c_i branch-target class, and d_i dynamic status. The raw mnemonic and address remain in the record for auditability but are not treated as a high-cardinality semantic target. A missing value is encoded as **__MISSING__**; a value not present in the training support is encoded as **__UNKNOWN__**. This makes the missing-field policy an explicit model input, rather than an implicit zero or an exception.

3.1.4 Observation semantics and protected evidence

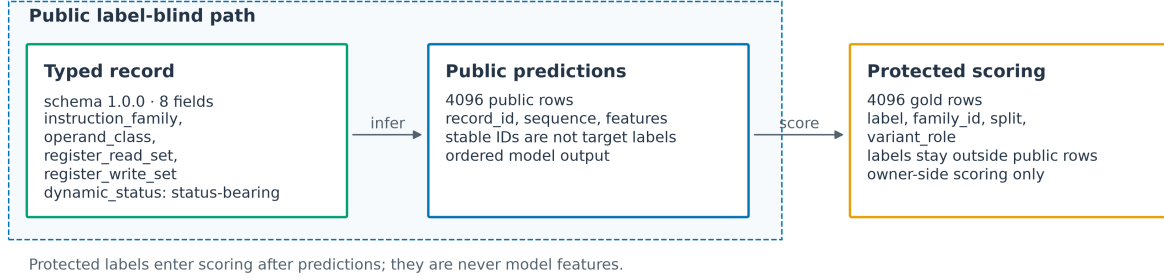
The observation contract makes absence and protection first-class data states. An observed categorical value is different from a field that an action did not produce; a value outside the training vocabulary is different from either of those; and an instruction, target, or capability outside the declared contract is not a measured numeric outcome. The serialized tokens **__MISSING__** and **__UNKNOWN__** preserve the first two distinctions through feature construction, while structured **unsupported** and **unavailable** statuses preserve the latter two. A genuine numeric zero remains valid when, and only when, a producer receipt measures zero. This prevents a convenient sentinel from silently changing the estimand or making a capability gap appear to be a failed method.

The same principle governs the replay interface. Public rows expose only the label-blind fields declared by the replication manifest; predictions return with stable identifiers; and gold labels remain in a protected scoring surface.

Protocol and artifact hashes connect these surfaces without turning protected labels into model inputs. The boundary is provenance-preserving, not a cryptographic claim about every possible side channel. It establishes what the model was permitted to observe and what the scorer was permitted to know.

Observation semantics and protected scoring

Typed values, missingness, unseen categories and protected labels have distinct representations.



Exact status table for the observation contract

category	serialized representation	interpretation boundary
observed	typed categorical value	measured at the declared observation grain
missing	__MISSING__	field not supplied; distinct from a measured zero
unseen	__UNKNOWN__	value outside training support; distinct from missing
unsupported / unavailable	structured status	outside semantic or local capability contract; no numeric imputation
measured zero	0 from receipt	numeric zero is retained only when the producer measured zero
protected label	outside public rows	owner-side scoring boundary; never a model feature

Protocol active-assembly-open-roadmap-v2-2026-10-13 · hash d394c9264fb6 · replay receipt implemented_local. Receipts provide provenance without revealing protected labels.

Figure 3: The observation-semantics diagram distinguishes the typed schema’s 8 fields, the dynamic missing token **MISSING**, and the unseen token **UNKNOWN** from a measured numeric zero. It also keeps unsupported instruction effects and unavailable compiler or target capabilities as explicit statuses rather than numeric values. The public replay surface exposes 3 label-blind fields, while 5 forbidden fields remain behind the protected scoring boundary. This is a schema and provenance visual: it documents how absence and protection are represented before inference, not how well a model performs. The protocol is active-assembly-open-roadmap-v2-2026-10-13 with hash prefix d394c9264fb6. It does not establish complete semantic coverage, recoverability of protected labels, or universal identifiability.

fig. 3 is therefore a schema and evidence-boundary visual rather than a performance result. It is generated from the typed observation, research-protocol, and replication receipts, and its exact status table makes the semantics inspectable in grayscale and without relying on color. The diagram does not assert that the current observation channel is complete, that protected labels are recoverable, or that agreement under the finite channel entails unrestricted program equivalence.

The typed records are evaluated as a companion to the collapsed channel across 10 complete labelled fixtures. For a training-only support V_f for field f , a smoothed field factor is

$$\hat{p}_f(v \mid s) = \frac{n_{f,v,s} + \alpha}{\sum_{v' \in V_f} n_{f,v',s} + |V_f|\alpha}, \quad (5)$$

and the factorized evidence score is

$$\ell_{\text{fact}}(s \mid \mathbf{x}_i) = \sum_{f \in \mathcal{F}} \log \hat{p}_f(x_{i,f} \mid s). \quad (6)$$

The hierarchical companion groups fields into instruction semantics, register context, control flow, and dynamic context. It averages log scores within each active group and then averages groups before normalizing over states:

$$\ell_{\text{hier}}(s \mid \mathbf{x}_i) = \frac{1}{|\mathcal{G}_i|} \sum_{g \in \mathcal{G}_i} \frac{1}{|\mathcal{F}_g|} \sum_{f \in \mathcal{F}_g} \log \hat{p}_f(x_{i,f} \mid s). \quad (7)$$

This group-balanced score is intentionally called a composite score: it is not presented as proof that the fields are conditionally independent, nor as a normalized joint distribution over all assembly records. The existing six-category likelihood remains the deliberately collapsed reference. The complete schema is visualized in fig. 4, while the fixture-level comparison, field-family ablation, and masked-memory sensitivity are shown in fig. 5.

Typed observation record: x86_accumulator

Text preserves categorical values; shaded row bands identify evidence families. An em dash is missing data, not zero.

	step 1 mov 0x1000	step 2 add 0x1005	step 3 shl 0x1008	step 4 sub 0x100a
instruction family	data	arithmetic	bitwise	arithmetic
operand class	mixed	mixed	mixed	mixed
registers read	∅	eax	eax	eax
registers written	eax	eax+rflags	eax+rflags	eax+rflags
flag effect	none	write	write	write
memory mode	none	none	none	none
branch target	none	none	none	none
dynamic status	executed	executed	executed	executed

Static fields come from Capstone detail; dynamic status comes from bounded Unicorn execution.

Figure 4: Typed observation record for the canonical fixture. Text cells preserve the instruction family, operand class, Capstone register read/write sets, flag effect, memory addressing mode, branch target class, and bounded dynamic status; shaded bands identify field families. An em dash means missing data rather than zero, and the table is categorical rather than a numeric heatmap [Intel Corporation, 2026, Capstone Engine Project, 2026, Unicorn Engine Project, 2026].

The factorized evaluation uses 4 field groups and 4 corresponding leave-one-group-out ablations, while holding out complete fixtures. For each held-out fixture f_k , every field support, conditional factor, transition count, and initial-state count is fit on $\mathcal{F}_{-k}$; the excluded record is then decoded once. The artifact is `./output/data/factorized_observation_model.json`. The resulting comparison is explicitly diagnostic: collapsed mean accuracy is 0.667, static typed factorization is 0.520, full factorization is 0.520, and the hierarchical composite score is 0.361. Thus the full typed score changes the collapsed reference by -0.147 and changes the static typed ablation by 0.000 on this tiny corpus. Masking the memory-addressing field yields mean accuracy 0.562 (0.042 relative to full factorization), a useful null result because the labelled fixtures contain little memory-mode variation.

3.1.5 Fixture-design coverage

The typed comparison is now evaluated on an expanded transparent corpus rather than on register-only straight-line variants. The corpus contains 10 complete fixtures. A source-bound coverage inventory records 2 fixture with explicit memory addressing, 1 fixture with a statically decoded but unexecuted instruction, and 8 fixtures with repeated authored phase labels. The inventory also records 5 instruction families, 2 non-null memory addressing mode(s), and 3 non-null branch-target class(es). These counts are derived from the real Capstone and bounded Unicorn producer path; they describe fixture design, not independent binary samples.

Typed evidence is an evaluated companion, not an automatic improvement

Each model is fit on the other fixtures; missing values map to `__MISSING__` and unseen training values to `__UNKNOWN__`.

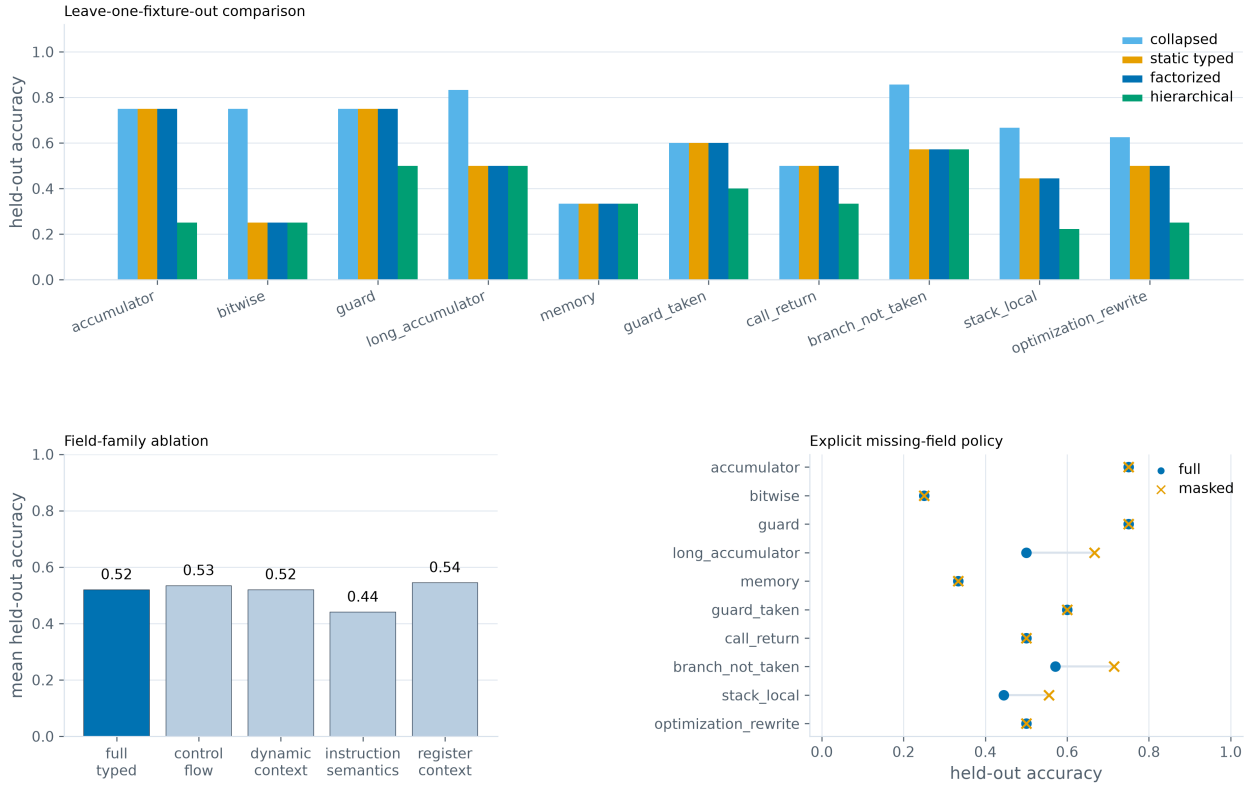


Figure 5: Leave-one-fixture-out comparison of the collapsed categorical HMM with static typed, full factorized, and hierarchical composite observation scores. The upper panel reports held-out accuracy per fixture, the lower-left panel removes one typed field family at a time, and the lower-right panel pairs full and memory-masked accuracy by fixture under the explicit missing-field policy. Every fit excludes the displayed fixture; no bar is a population estimate. The hierarchical score balances evidence groups and is not interpreted as a normalized joint likelihood [Rabiner, 1989, Gelman et al., 2013].

For fixture k , define the design vector

$$\mathbf{c}_k = (N_k, E_k, R_k, M_k, B_k, U_k), \quad (8)$$

where N_k is the number of statically decoded instructions, E_k is the number of distinct dynamically executed addresses, R_k is the number of contiguous authored phase runs, M_k is the number of static memory-operand instructions, B_k is the number of observed non-null branch-target classes, and U_k is the number of typed instruction families. The coverage artifact is `../output/data/corpus_coverage.json`; fig. 6 visualizes the quantitative N_k/E_k distinction and the categorical design flags. The artifact is a promotion gate for fixture diversity, not a semantic score: it cannot show that a field is intrinsic to program meaning or that 10 toy programs represent arbitrary binaries.

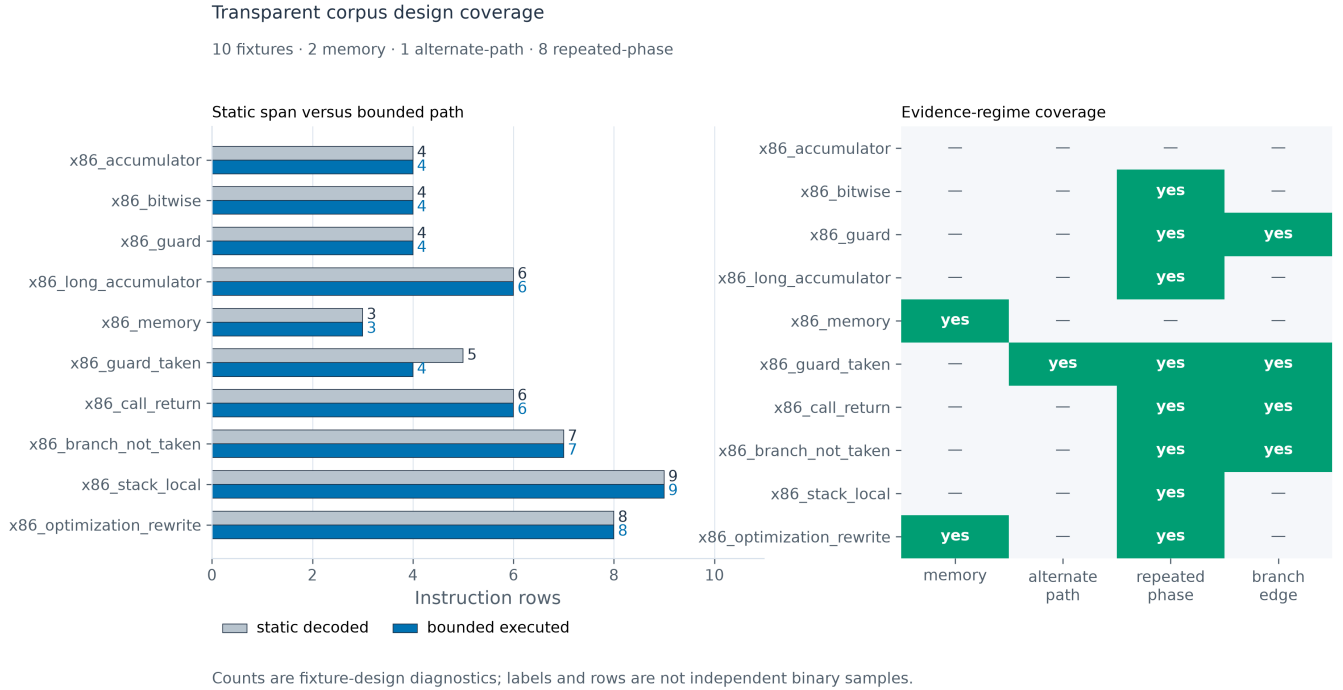


Figure 6: Transparent fixture-design coverage across the 10-fixture corpus. The left panel separates static decoded span from bounded executed path length; the right panel records the presence of memory operands, alternate dynamic paths, repeated authored phases, and direct branch edges. The corpus contains 2 memory fixture(s), 1 alternate-path fixture(s), and 8 repeated-phase fixture(s). These are design diagnostics, not independent binary samples.

3.1.6 Bounded transfer functions and control-flow formalism

The categorical map is supplemented by a small executable semantics layer. Capstone detail provides register read/write sets and direct operand spelling. A bounded transfer function executes the documented immediate-register, arithmetic, bitwise, shift, comparison, and conditional-branch subset over the dynamic path. The report is emitted as `../output/data/symbolic_execution.json`.

For the selected fixture, 4 instructions are covered by the transfer-function subset and 0 executed instructions are unsupported; the symbolic final value agrees with Unicorn (yes). Unsupported executed instructions fail closed rather than receiving a guessed effect. This is an auditable explanation layer, not a complete x86 lifter, symbolic solver, decompiler, or claim of semantic program understanding [Intel Corporation, 2026, Capstone Engine Project, 2026, Unicorn Engine Project, 2026, Dasgupta et al., 2019]. The distinction is substantive: complete formal x86-64 semantics are a separate research object, whereas this package deliberately verifies only a small transfer-function subset and fails closed outside it [Dasgupta et al., 2019].

Let q_i be the concrete register state before instruction i and T_i the transfer function selected by its decoded mnemonic and operands. For an executed supported instruction, the bounded execution is

$$q_{i+1} = T_i(q_i), \quad e_{i+1} = \Psi_i(e_i, \text{operands}_i), \quad (9)$$

where e_i is the parallel readable symbolic expression. The independent Unicorn check tests the concrete terminal projection,

$$\pi_{RAX}(q_N^{\text{symbolic}}) = q_N^{\text{Unicorn}}, \quad (10)$$

not equality of the textual expression.

The stronger instrument contract now compares the bounded register projection at every executed step. Let u_i^- and u_i^+ be the pre- and post-step Unicorn snapshots aligned to the decoded address, and let $\hat{q}_i^+ = T_i(\hat{q}_i^-)$ be the transfer-function update. The stepwise RAX endpoint is

$$I_i^{RAX} = \mathbb{1}[\pi_{RAX}(\hat{q}_i^-) = \pi_{RAX}(u_i^-) \wedge \pi_{RAX}(\hat{q}_i^+) = \pi_{RAX}(u_i^+)], \quad (11)$$

and the report records 4 agreements out of 4 comparable executed steps. For architectural flags, the interpreter compares only bits it defines for a particular operation: carry (CF), zero (ZF), sign (SF), and overflow (OF). It records 12 agreements out of 12 defined bit comparisons. An undefined or preserved flag is not silently treated as a match. These are trace-refinement checks for a declared subset, not a proof of the complete x86-64 transition relation [Intel Corporation, 2026, Dasgupta et al., 2019]. The stepwise refinements are visualized in fig. 7.

Bounded transfer functions refine the observed execution trace

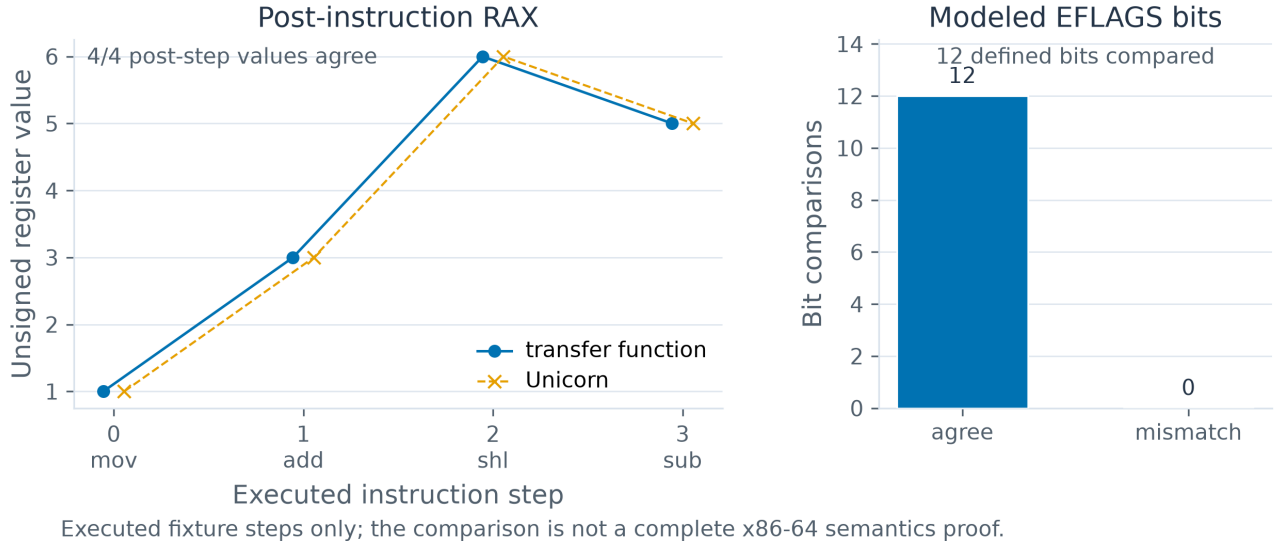


Figure 7: Stepwise trace-refinement check for the canonical bounded x86-64 execution. The left panel overlays post-instruction RAX values from the closed-world transfer functions and real Unicorn, with one marker per executed instruction; the right panel counts agreement and mismatch for only the EFLAGS bits defined by the transfer functions. Static instructions not visited by Unicorn are omitted. Exact stepwise agreement strengthens the terminal-value check but remains a fixture-bounded instrument validation, not a complete x86-64 semantics theorem [Intel Corporation, 2026, Dasgupta et al., 2019, Unicorn Engine Project, 2026].

The coverage matrix (fig. 8) should be interpreted as a scope ledger. A check mark in the Lean column is evidence about the checked formal fragment; a Capstone or Unicorn mark is evidence about the corresponding adapter call on the recorded fixture. These marks are not interchangeable and do not aggregate into a percentage of the entire x86-64 architecture. Textual statuses and hatching preserve the distinction between unsupported semantics, unavailable tooling, and a cell that has simply not yet been observed.

Declared semantic coverage by evidence layer

Operation-family matrix from the formal and adapter receipts; coverage status is categorical and does not imply complete x86-64 semantics.

✓ = declared or checked; unresolved/not observed = evidence not present; hatching = no numeric performance value

	declared	width	flags	memory	control	Lean	Capstone	Unicorn
moves / extensions	✓	✓	declared	not central	not central	unresolved	✓	✓
integer arithmetic	✓	✓	✓	not central	not central	✓	✓	✓
logic / comparisons	✓	✓	✓	not central	not central	unresolved	✓	not observed
shifts	✓	✓	✓	not central	not central	unresolved	✓	✓
control flow	✓	declared	✓	not central	✓	unresolved	✓	not observed
stack / memory	✓	declared	declared	✓	not central	unresolved	✓	not observed
outside contract	UNSUPPORTED	—	—	—	—	—	—	—

SIMD, floating point, privileged/system instructions, unmodeled exceptions, and unbounded aliasing; Instructions whose flag dependencies are outside the modeled definedness contract

Figure 8: The semantic coverage matrix records 29 declared instruction names, 71 kernel-checked theorem entries, and 24 formal correspondence step rows across operation families and evidence layers. Columns distinguish declaration, operand-width handling, flag effects, bounded memory, control flow, Lean coverage, Capstone availability, and Unicorn cross-checks. Check marks therefore mean a declared or observed capability at the stated layer; unresolved, not-observed, and unsupported cells are written as text and hatching rather than zero. The matrix is a scope receipt for a user-mode integer subset and its current transparent fixtures. It is not a complete x86-64 ISA table, a proof of all adapter behavior, or a performance result.

3.1.7 Mechanically checked closed-world semantics

The executable transfer functions are complemented by a small, independent operational-semantics fragment in Lean. Its purpose is not to relabel the Capstone trace as a proof of x86-64 correctness; it is to make the algebraic equivalence witnesses used by the quotient-state program explicit and kernel checked. The formal state is a modular word $q \in \mathbb{W}_{64} = \{0, \dots, 2^{64} - 1\}$, and the closed operation set is **mov**, **add**, **sub**, **inc**, and **nop**. The formal step function is

$$\begin{aligned}
 \text{step}(\text{mov } v, q) &= v, & \text{step}(\text{add } v, q) &= q + v \pmod{2^{64}}, \\
 \text{step}(\text{sub } v, q) &= q - v \pmod{2^{64}}, & \text{step}(\text{inc}, q) &= q + 1 \pmod{2^{64}}, \\
 \text{step}(\text{nop}, q) &= q.
 \end{aligned} \tag{12}$$

For a finite program P , define its observational relation over this formal domain by

$$P \equiv_{64} Q \iff \forall q \in \mathbb{W}_{64}, \text{run}(P, q) = \text{run}(Q, q). \tag{13}$$

The checked receipt is `../output/data/formal_verification.json`, visualized in fig. 9: Lean toolchain `leanprover/lean4:v4.35.0` builds 71 named theorems and reports status `locally_verified`. The current theorem manifest proves two **inc** operations equal one addition by two, proves that **nop** is neutral in a finite program, proves two constant-addition programs equivalent under every formal initial word, proves a changed subtraction is *not* equivalent by an explicit initial-state witness, and proves that appending a **nop** preserves the formal run. The trusted base is the Lean kernel and its imported core/standard library; no external solver output is accepted as a proof certificate. The generated bridge also serializes a transparent boundary program from the Python reference path and checks its expected modular `Word64` result in Lean. This shared fixture is a checked cross-language anchor, while the broader Python/Capstone/Unicorn correspondence remains computational evidence. This is a mechanically checked algebraic prerequisite for the local quotient protocol, not a theorem about Capstone, Unicorn, x86 flags, memory, exceptions, compiler correctness, or all assembly programs. Lean 4’s scholarly account describes the system as both an extensible theorem prover and a programming language [de Moura and Ullrich, 2021]; here, only the kernel checked receipt for the declared local fragment is promoted.

Formal verification: checked fragment and evidence boundary

Lean toolchain leanprover/lean4:v4.35.0 · 71 declared theorems · endpoint: locally_verified

Evidence status ledger

The receipt distinguishes checked results from absent evidence.

evidence status	receipt value	interpretation
proof-backed	71	Lean kernel checked
cross-language bridge	1 fixture	generated trace: bounded scope
computational-only	not recorded	not promoted by this receipt
falsified candidate	not recorded	see theorem status map
unresolved candidate	not recorded	see theorem status map
unsupported / excluded	1	outside declared fragment
underpowered	1	not a positive result

Checked fragment

- mov immediate
- add immediate
- sub immediate
- inc
- cmp (exact sub flags with no writeback)
- and/or/xor logic flags with AF undefined-on-logic
- width-aware moves (8/16-bit preserve upper bits, 32-bit zero-extends, movzx/movsx extension)
- push/pop over the bounded little-endian stack window with explicit boundExceeded failure
- jcc conditional branches over the flag definedness map with fail-closed exceptional status on undefined flag reads
- width-aware register writes
- explicit flag definedness and carry preservation
- bounded memory-state fields
- nop

Formal boundary

state: explicit 16-register normalized bounded machine-state schema; relation: universal final accumulator equality over the formal Word64 domain

Unsupported ISA effects, complete x86-64 semantics, and population-level claims remain outside this receipt.

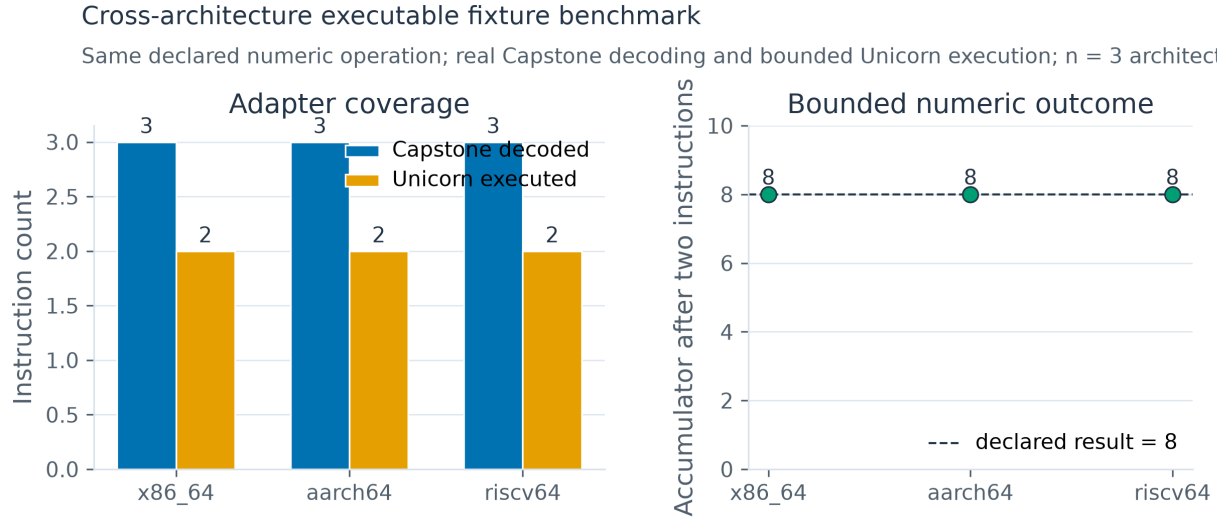
Kernel-checked rows are not a complete x86-64 semantics proof; computational bridge evidence and absent theorem statuses remain explicitly separated.

Figure 9: Formal verification receipt for the declared Word64 fragment under toolchain leanprover/lean4:v4.35.0. The upper status ledger separates kernel-checked theorem counts, bounded fixture-bridge evidence, and candidate outcomes not recorded in this receipt. The lower panel lists the checked operations and the normalized machine-state and relation boundary. Counts describe the local proof artifact; they are not confidence intervals or independent experiments. The receipt does not certify every Python adapter, unsupported ISA effects, or complete x86-64 semantics [Lean Community, 2026, Dasgupta et al., 2019].

3.1.8 Cross-architecture adapter falsification surface

The first architecture extension is deliberately a runtime adapter benchmark, not a portability claim. The same declared numeric operation is encoded as transparent bytes for x86-64, AArch64, and RISC-V64. Capstone decodes three instructions in each fixture; Unicorn executes exactly the two non-returning instructions; and the final accumulator is compared with the declared value. The receipt `../output/data/cross_architecture_fixtures.json` records the bytes, decoder mnemonics, API provenance, bounded instruction count, and outcome for all 3 architectures. All bounded executions passed (**yes**) and the common numeric result is 8; the benchmark is visualized in fig. 10.

The adapter contract is intentionally weaker than semantic equivalence. It does not compare flags, exception behavior, calling conventions, memory effects, ABI obligations, compiler transformations, or complete instruction semantics across ISAs. Its scientific value is adversarial at the software boundary: a real decoder/emulator path must execute before an architecture is counted, and a stale hand-written result cannot satisfy the receipt. The benchmark is therefore a falsification surface for adapter drift and a starting point for source-family-disjoint cross-compiler work, not evidence of cross-architecture latent-state invariance.



The shared result is a local adapter check, not a cross-ISA semantic-equivalence theorem.

Figure 10: Cross-architecture executable fixture benchmark for the same declared numeric accumulator operation. The left panel reports the three Capstone-decoded instructions and the two non-returning instructions actually executed by Unicorn for x86-64, AArch64, and RISC-V64; the right panel reports the resulting accumulator value and the declared value of 8. All rows pass the bounded adapter contract. The shared numeric result is a local runtime check, not a cross-ISA semantic equivalence theorem or compiler-generalization result [Capstone Engine Project, 2026, Unicorn Engine Project, 2026].

The static direct-branch partition produces 1 basic blocks and 0 CFG edges in the current bundle. A target outside the decoded span remains an explicit external edge, so the representation records the observed boundary without overstating it as whole-binary CFG recovery.

More formally, let each decoded instruction be

$$u_i = (a_i, \ell_i, m_i, x_i, R_i, W_i), \quad (14)$$

where a_i and ℓ_i are its address and byte length, m_i is the mnemonic, x_i is the operand string, and R_i/W_i are Capstone-derived register read/write sets. Let L contain instruction zero, every in-range direct branch target, and the instruction immediately after each branch or return. Sorting $L = (\lambda_0, \dots, \lambda_{K-1})$ yields the bounded basic-block partition

$$\mathcal{B}_j = \{u_i : \lambda_j \leq i < \lambda_{j+1}\}, \quad \lambda_K = N, \\ \bigcup_{j=0}^{K-1} \mathcal{B}_j = \{u_0, \dots, u_{N-1}\}. \quad (15)$$

An edge $(\mathcal{B}_j, \mathcal{B}_k, \tau)$ is emitted for a direct jump, conditional taken branch, or fall-through successor; an out-of-span target is retained as an **external_*** endpoint. This is a finite CFG representation of the fixture’s decoded span, not whole-program reachability.

For the separate leakage-safe estimation path, let f_k be the held-out fixture and $\mathcal{F}_{-k}$ the remaining labelled fixtures. With positive smoothing concentrations $\alpha_A, \alpha_B, \alpha_D$, the fitted categorical parameters are

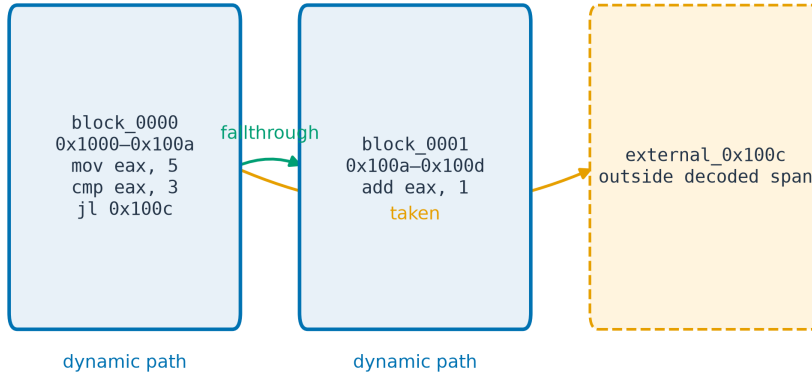
$$A_{o,s}^{(-k)} = \frac{n_{o,s}^{(-k)} + \alpha_A}{\sum_{o'} n_{o',s}^{(-k)} + |\mathcal{O}| \alpha_A}, \\ B_{s',s}^{(-k)} = \frac{n_{s',s}^{(-k)} + \alpha_B}{\sum_r n_{r,s}^{(-k)} + |\mathcal{S}| \alpha_B}, \\ D_s^{(-k)} = \frac{n_s^{(-k)} + \alpha_D}{\sum_r n_r^{(-k)} + |\mathcal{S}| \alpha_D}. \quad (16)$$

The superscript $(-k)$ is an exclusion contract: no observation, phase label, or transition count from f_k enters its fitted tensors. The resulting model is then scored once on f_k and written to `../output/data/held_out_model.json`. This is supervised fixture-level estimation, not unsupervised phase discovery.

The branch-bearing CFG evidence is retained in `../output/data/cfg_evidence.json` and visualized in fig. 11. It makes the distinction between a decoded direct branch, a fall-through successor, and an out-of-span target inspectable without importing a larger reverse-engineering framework.

Capstone-derived blocks and direct edges; blue border = observed by bounded Unicorn execution; amber dashed node = explicit external target.

Static CFG evidence for x86_guard



edge colors: green = fall-through · amber = branch-taken possibility

Figure 11: Static control-flow evidence for the branch-bearing x86_guard fixture. Each block preserves its decoded address range and Capstone mnemonic sequence; green fall-through and amber taken-branch edges are direct-target relations, blue borders mark blocks observed on the bounded Unicorn path, and the dashed amber external node records an out-of-span target. This is a finite fixture CFG, not whole-binary reachability or decompilation.

3.1.9 Memory operands and dynamic access provenance

Register traces alone are insufficient for an assembly-to-inference interface: an instruction can read or write memory without changing the final accumulator, and a static memory operand does not establish that the access occurred. The method therefore includes a transparent `x86_memory` fixture in the expanded labelled corpus. Capstone records the base register, index register, scale, displacement, and operand width for every memory operand; Unicorn memory hooks record the access type, effective address, width, value, triggering instruction address, and dynamic event index. The memory-evidence artifact remains a separately named provenance projection even though the same fixture now participates in the typed leave-one-fixture-out evaluation; this keeps the store/load join inspectable without silently changing the meaning of the collapsed channel.

The static operand relation is written as

$$m_i = (b_i, j_i, \sigma_i, d_i, w_i), \quad \hat{a}_i = R[b_i] + \sigma_i R[j_i] + d_i, \quad (17)$$

where b_i and j_i are the base and index registers (either may be absent), σ_i is the scale, d_i is the signed displacement, and w_i is the operand width. A dynamic memory event is

$$\mu_k = (k, \rho_k, a_k, w_k, v_k, i_k), \quad (18)$$

with event order k , access relation $\rho_k \in \{\text{read}, \text{write}\}$, effective address a_k , width w_k , observed value v_k , and triggering instruction index i_k . The contract joins the two streams only when i_k has a declared memory operand and the observed address equals the bounded Unicorn register state plus the decoded displacement. It does not perform alias analysis, recover heap objects, or infer memory meaning from an address alone.

The current provenance artifact is `../output/data/memory_evidence.json`, visualized in fig. 12. It contains 2 static memory instructions and 2 dynamic events: 1 write and 1 read event(s) at 1 distinct effective address(es). The store and load observe the same fixture value, providing a minimal end-to-end check that the static operand, effective address, dynamic hook, and terminal register result agree. This is an instrumentation demonstration, not evidence of general memory semantics.

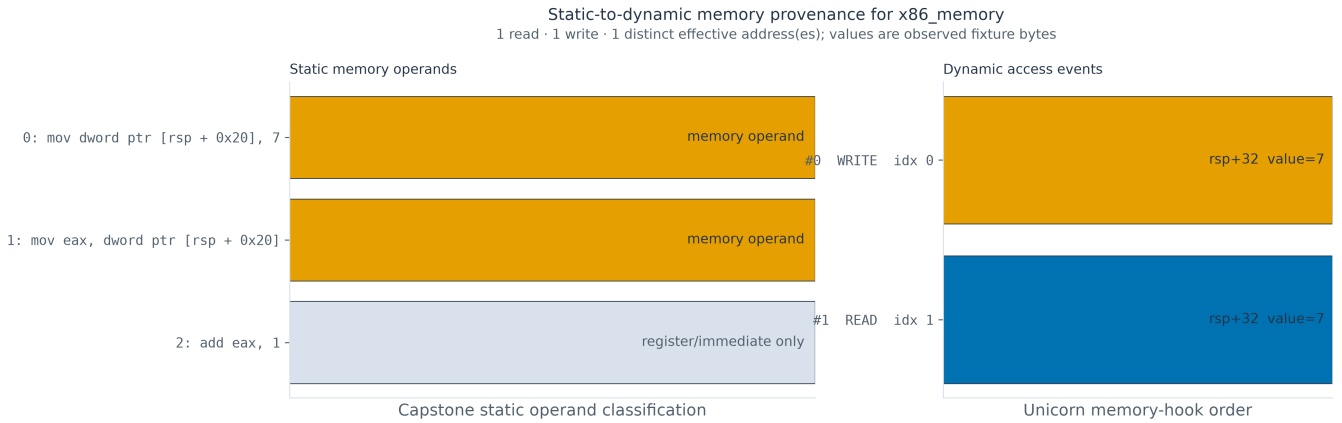


Figure 12: Static-to-dynamic memory provenance for the transparent `x86_memory` fixture. The left panel marks which Capstone-decoded instructions contain explicit memory operands; the right panel orders the real Unicorn memory-hook events and reports access type, triggering instruction, stack-relative effective address, and observed value. The write/read pair demonstrates a bounded store-load join, not general heap recovery, alias analysis, or arbitrary-binary tracing.

The native RxInfer layer is invoked through `julia --project` and reads the same trace bundle. The current model maps observation IDs into numeric event codes and runs a conjugate Gaussian observation model with `free_energy=true`. Every exposed `result.free_energy` value is written to `output/rxinfer/free_energy_trace.json` 1 with data and configuration hashes, following the RxInfer static-inference interface [ReactiveBayes, 2026b]. The Python-to-RxInfer integration is a native Julia CLI call because the RxInfer documentation identifies the server/client route as third-party and not maintained by the core team [ReactiveBayes, 2026a]. This Gaussian trajectory is an

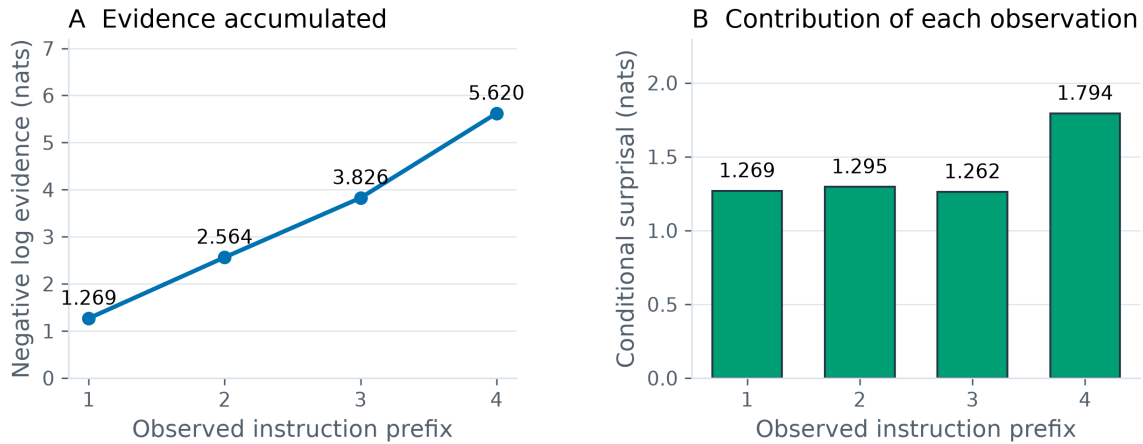
explicitly retained numeric event-code diagnostic. The same package now also runs a validated, namespaced categorical companion, `categorical phase hmm`, over one-hot observation categories and fixed left-to-right latent phases. Its diagnostics are written to `../output/rxinfer_categorical/categorical_diagnostics.json` and contain 8 finite values, with final value 5.619808. The fixed-chain iteration values are already stationary after the first sweep. The prefix-evidence and conditional-surprisal panels in fig. 13 instead show native inference on successively longer observed sequences, checked against an independent forward calculation. They are evidence accumulation diagnostics, not learning or convergence curves [RxInfer contributors, 2026a,b]. The Gaussian and categorical free-energy values are not compared on an absolute scale: they are distinct factor graphs with different variational objectives.

The categorical companion makes the intended phase model explicit. For latent states z_t and categorical observations x_t ,

$$p(z_{1:T}, x_{1:T}) = p(z_1) \prod_{t=2}^T p(z_t | z_{t-1}) \prod_{t=1}^T p(x_t | z_t), \quad (19)$$

where the transition and emission matrices are recorded in the companion configuration artifact. RxInfer executes this model through its native `Categorical` and `DiscreteTransition` factors; the Julia package tests assert finite, correctly namespaced free-energy and posterior artifacts. The companion is therefore an implemented method path, while its fixed parameters and small fixture input remain insufficient for claims of learned program semantics.

Categorical GNN model: native evidence by observation prefix



Full-sequence check: 8 identical iteration values; final F = 5.620. Fixed parameters; no learning curve.

Figure 13: Native evidence accumulation under the fixed GNN categorical phase-HMM. Panel A shows separate one-iteration RxInfer runs on successive observation prefixes; panel B shows their differences, the conditional surprisal of each observation. The independent forward recursion verifies every prefix. This changes the conditioning data, not the parameters, and is not an optimization or learning curve. The retained full-sequence check records all values are identical to recorded precision with delta 0.000. The objective remains distinct from Gaussian event-code free energy and policy EFE [RxInfer contributors, 2026a,b].

3.2 Temporal, active, and evaluated inference

The identity-transition comparison filter carries the previous posterior forward unchanged and therefore omits state dynamics — implicitly assuming an identity transition over latent phases. The temporal model adds a left-to-right transition matrix over the 5 program phases, yielding a genuine discrete hidden Markov model. Forward filtering alternates a predict step (the transition matrix applied to the previous posterior) with a categorical likelihood update; Viterbi decoding recovers the most probable joint phase path; and forward-backward smoothing refines the per-step beliefs [Friston et al., 2017, Rabiner, 1989]. Contiguous program phases are then segmented directly from the smoothed beliefs.

The category label is the model input; the mnemonic/operand text is the auditable assembly evidence.

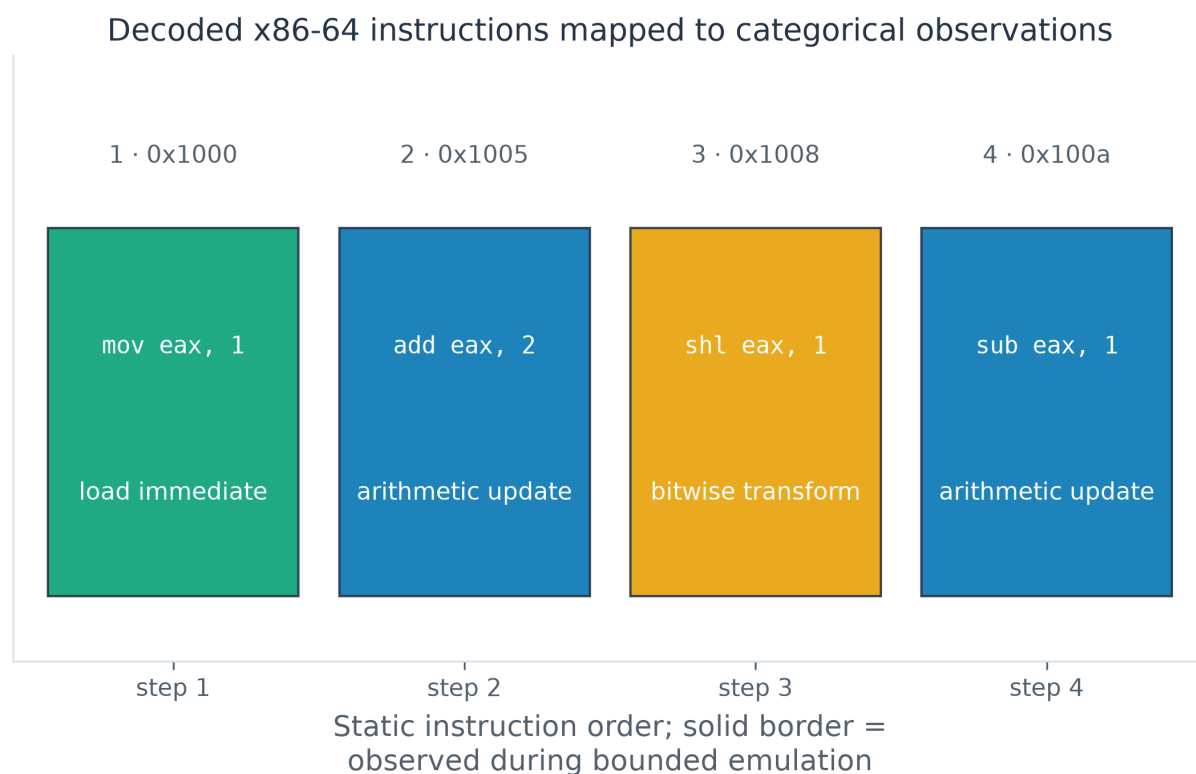


Figure 14: Source-bound assembly evidence map for the selected x86-64 fixture. Each card shows the decoded address, mnemonic/operands, and categorical channel consumed by the model; the instruction order is static, and a solid border marks an address observed during bounded Unicorn execution. Observation IDs are labels, not continuous measurements.

Filled circles and direct labels identify found adapters; x markers identify absent probes.



Figure 15: Package-provenance audit for the required Capstone and Unicorn adapters plus optional roadmap slots. Filled circles and direct labels mark packages found by the local probe; x markers mark packages not found. Required/optional status is printed beside each row. Optional rows are recorded as provenance and are not treated as evidence streams.

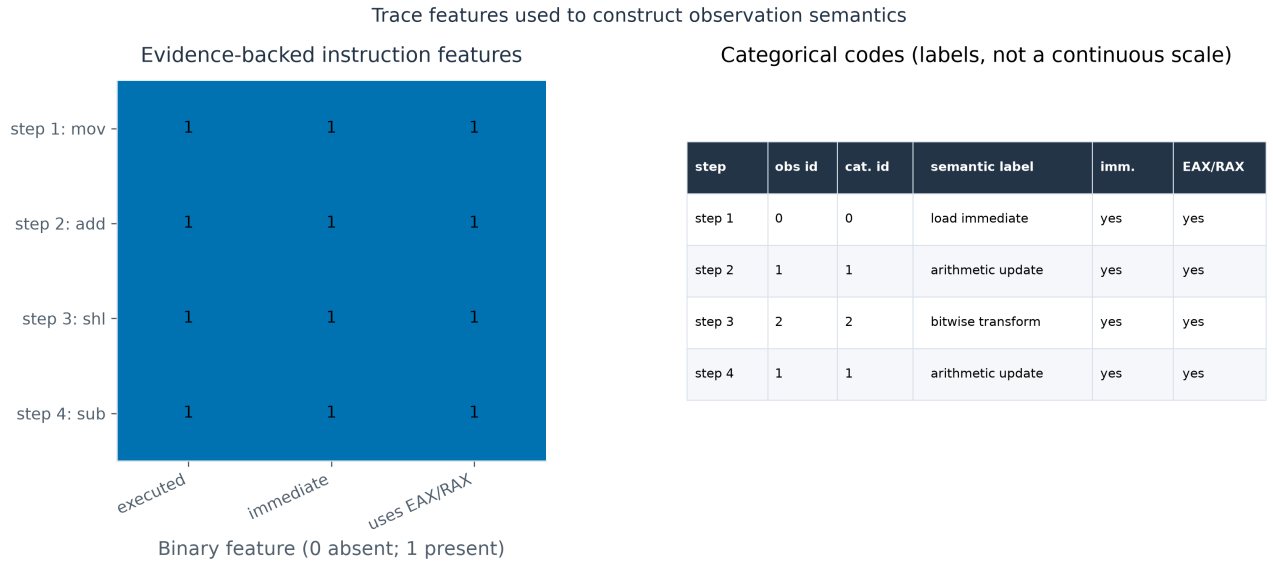


Figure 16: Trace-feature audit for every decoded instruction. The left panel contains binary flags for dynamic execution, an immediate operand, and an accumulator operand; the right table preserves step, raw observation ID, categorical code, semantic label, and the two operand flags. The codes are annotated as categories and must not be read as a continuous scale.

The complete Bayesian graph of this model is drawn in fig. 17: the hidden phase states form the left-to-right Markov backbone, each emits a categorical observation through the likelihood $\mathbf{A}$, the initial belief is fixed by the prior $\mathbf{D}$, and the active layer scores candidate observations through the expected free energy.

3.2.1 Observation-only parameter learning

The fixed matrices are complemented by an explicitly separated parameter-learning path. For each complete-fixture holdout, the training input is the sequence of observation-category IDs from the other fixtures; the hand-authored phase labels are not passed to fitting. This is an observation-only estimator over a declared ordered-state scaffold, not a discovery algorithm. Because categorical latent states are label-non-identifiable without a constraint, the fit is initialized from the inspectable reference model and retains its ordered phase names only as an architectural coordinate system. The held-out labels are introduced exactly once, in the scoring layer.

For observation sequence $o_{1:T}$, the E step computes smoothed responsibilities

$$\gamma_t(s) = P(s_t = s \mid o_{1:T}), \quad \xi_t(s', s) = P(s_t = s, s_{t+1} = s' \mid o_{1:T}), \quad (20)$$

using the scaled forward–backward recursions. With positive Dirichlet pseudocounts, the M step updates the column-stochastic emission and transition parameters as

$$\hat{A}_{o,s} = \frac{\alpha_A + \sum_t \mathbf{1}[o_t = o] \gamma_t(s)}{|\mathcal{O}| \alpha_A + \sum_t \gamma_t(s)}, \quad (21)$$

and, for the ordered support $\mathcal{E} = \{(s, s), (s+1, s)\}$,

$$\hat{B}_{s',s} \propto \epsilon + \mathbf{1}[(s', s) \in \mathcal{E}] \left(\alpha_B + \sum_t \xi_t(s', s) \right), \quad \hat{D}_s \propto \alpha_D + \gamma_1(s). \quad (22)$$

Here ϵ is a strictly positive transition floor, which keeps the shared log-space inference contract valid while making the self/next-state prior soft rather than a hard zero. Each training complement is iterated for at most 32 updates, and the fitted model is scored on the excluded fixture without further adaptation. The complete rows, fitted tensors, evidence histories, and fit-input contract are emitted in `../output/data/parameter_learning.json`. The estimator follows the incomplete-data maximum-likelihood logic of the EM algorithm [Dempster et al., 1977] and standard HMM forward–backward inference [Rabiner, 1989]; the implementation remains fixture-bound and does not turn the phase scaffold into learned semantics.

Robustness is not summarized by selecting the most favorable start. A fixed- penalty Dirichlet-MAP comparator is fit from the same observation-only inputs, and its held-out likelihood remains separate from the primary ordered-EM score. For each held-out sequence, a label-free posterior-predictive check simulates same-length categorical sequences and compares observation entropy, category support, and adjacent-change rate with central 95% simulation intervals. The model-selection ledger records that any reference choice uses training evidence only; held-out phase labels enter only after fitting. Non-convergence, weak local optima, and seed spread remain visible in the receipt rather than being filtered from the figure.

To make initialization sensitivity part of the estimand rather than an invisible implementation choice, each holdout is fit from five frozen starts: the reference parameterization, a uniform parameterization, and three deterministic random starts with seeds recorded in the receipt. Let $\mathcal{J}$ denote these starts and let $\hat{y}_{h,i}$ be the held-out sequence predicted for holdout h from start i . We retain the within-holdout accuracy range

$$R_h = \max_{i \in \mathcal{J}} \text{Acc}(\hat{y}_{h,i}, y_h) - \min_{i \in \mathcal{J}} \text{Acc}(\hat{y}_{h,i}, y_h), \quad (23)$$

and the mean pairwise sequence agreement across starts. These are robustness diagnostics, not uncertainty intervals over fixtures: a large R_h or low agreement is evidence that the latent fit is initialization-sensitive. Start selection is not performed using held-out labels, and every start—including weak or unstable fits—is retained in `../output/data/parameter_learning.json`.

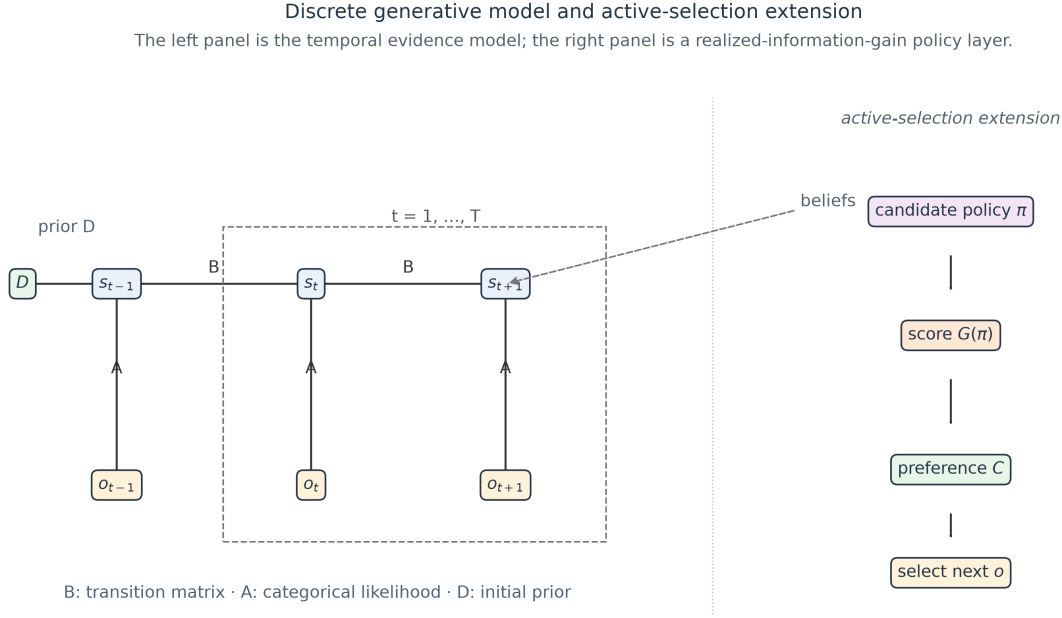


Figure 17: Conceptual graph of the discrete generative model and its active-selection extension. The left side shows hidden phases linked by transition B and emitting categorical observations through likelihood A from prior D ; the right side shows candidate policy π , score $G(\pi)$, preference C , and the selected next observation. This is a deterministic schematic of source-controlled model constants, not an empirical network diagram.

3.3 Formal description

Definition 1 (Discrete generative model). A discrete temporal generative model is the triple (D, B, A) where D is a prior over the hidden phase s_1 , B is a column-stochastic transition matrix $B[s', s] = P(s_{t+1} = s' \mid s_t = s)$, and A is a column-stochastic likelihood $A[o, s] = P(o_t = o \mid s_t = s)$. An observation sequence $o_{1:T}$ arises from the phases following the left-to-right Markov chain, each phase emitting one observation.

Definition 2 (Retrospective ordering score). For an already decoded candidate category o , the ordering score is the negative sum of its realized phase-entropy reduction and a log-probability preference $\ln r(o)$. The diagnostic selects the candidate minimizing that score. A flat probability preference r makes the pragmatic term constant across candidates. This retrospective preference is distinct from the dimensionless utility C in the canonical prospective sensor POMDP specified above.

Definition 3 (Log marginal likelihood (model evidence)). The model evidence of the observed traces under a generative model is the log marginal likelihood $\sum_t \ln P(o_t \mid o_{1:t-1})$, where each conditional is the forward-filter normalizer at step t .

Proposition 1 (Forward-filter normalizer is the evidence). At each step the forward filter’s normalizer equals the conditional marginal $P(o_t \mid o_{1:t-1})$, so the cumulative log of the normalizers is exactly the model’s log marginal likelihood, as defined above.

Proposition 2 (Uniform transitions collapse temporal to memoryless). When the transition matrix is uniform, $B @ \text{posterior}$ is a uniform vector, so the temporal forward update reduces exactly to the per-step memoryless posterior $P(s_t \mid o_t) \propto A[o_t]$; the temporal and memoryless decoders then coincide.

The companion heatmaps in fig. 18, fig. 23, fig. 24, fig. 25, fig. 26, and fig. 27 expose the same filtering contract across the labelled fixtures. They are supporting diagnostics rather than additional independent experiments: row labels are observation IDs, colors are posterior probabilities, and the hand-authored phase labels are not discovered from the bytes. Four further panels make the expanded fixture design explicit: the call/ret panel (fig. 19) shows an in-span call and **ret** roundtrip whose return address is pushed and popped inside the decoded span; the not-taken-guard panel (fig. 20) holds the guard phase across consecutive comparisons whose conditional branches are statically decoded but not taken; the stack-local panel (fig. 21) interleaves accumulator arithmetic with bounded stack-local state in repeated phase runs; and the optimization-rewrite panel (fig. 22) holds the add/inc/lea optimization-sensitive motif,

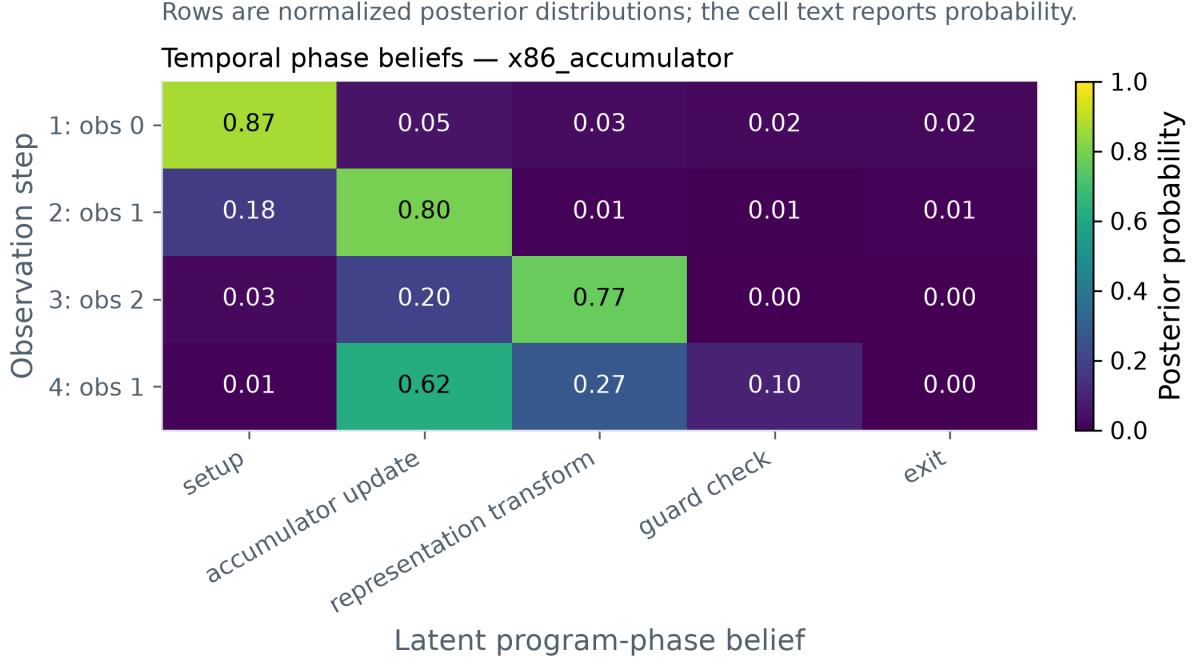


Figure 18: Temporal forward-filter beliefs for the x86_accumulator fixture. Each of the 4 rows is a normalized posterior over 5 hand-authored program phases after one observation; cell labels expose the probability rather than hiding it in color alone.

in which the `lea` rewrite lands in the residual `other` observation category even though its latent phase is accumulator update — a designed ambiguity retained as data, not smoothed away. The earlier two expanded panels retain a bounded stack-memory operand and a taken branch whose static fall-through instruction is not executed.

The retrospective ordering heuristic defined above scores each already decoded candidate by an epistemic term and a pragmatic term (a log-probability preference over observation categories), then selects the candidate with the lowest score [Friston et al., 2017]. This yields a category-informed instruction ordering compared with left-to-right scanning. Its acquisition boundary differs from that of the prospective sensor Agent specified above.

We retain two explicitly named epistemic quantities. The primary corpus policy uses the *realized* information gain of the category already decoded for a candidate. Its predictive companion weights that entropy reduction by $P(o)=A[o,:]\mathbf{D}$, the prior predictive probability of the category. Therefore

$$I(S; O) = \sum_o P(o) [H(D) - H(P(S | o))], \quad (24)$$

and the per-category weighted terms sum to the one-step expected information gain 0.667 nats for the authored model. This predictive quantity is non-negative in aggregate and provides an honest calibration diagnostic. It does not create an unmodelled stochastic sensor: candidate categories are still decoded before selection, so the active comparison remains a bounded demonstrator rather than a claim of deployment- ready active binary acquisition.

3.3.1 Action-conditioned sensing

The next active-inference tranche changes the information-acquisition contract. Rather than scoring a category that is already visible, a sensing action $\mathbf{a}=(\mathbf{t}, \mathbf{f})$ requests field $\mathbf{f}$ from instruction step $\mathbf{t}$. The action policy sees only a training-only field likelihood and the current phase belief before it chooses. The held-out typed value is revealed by the bounded fixture environment only after the choice. For a field support $\mathbf{V}_\mathbf{f}$, the predictive outcome distribution is

$$P(v | b_t, a) = \sum_s \hat{p}_f(v | s) b_t(s), \quad (25)$$

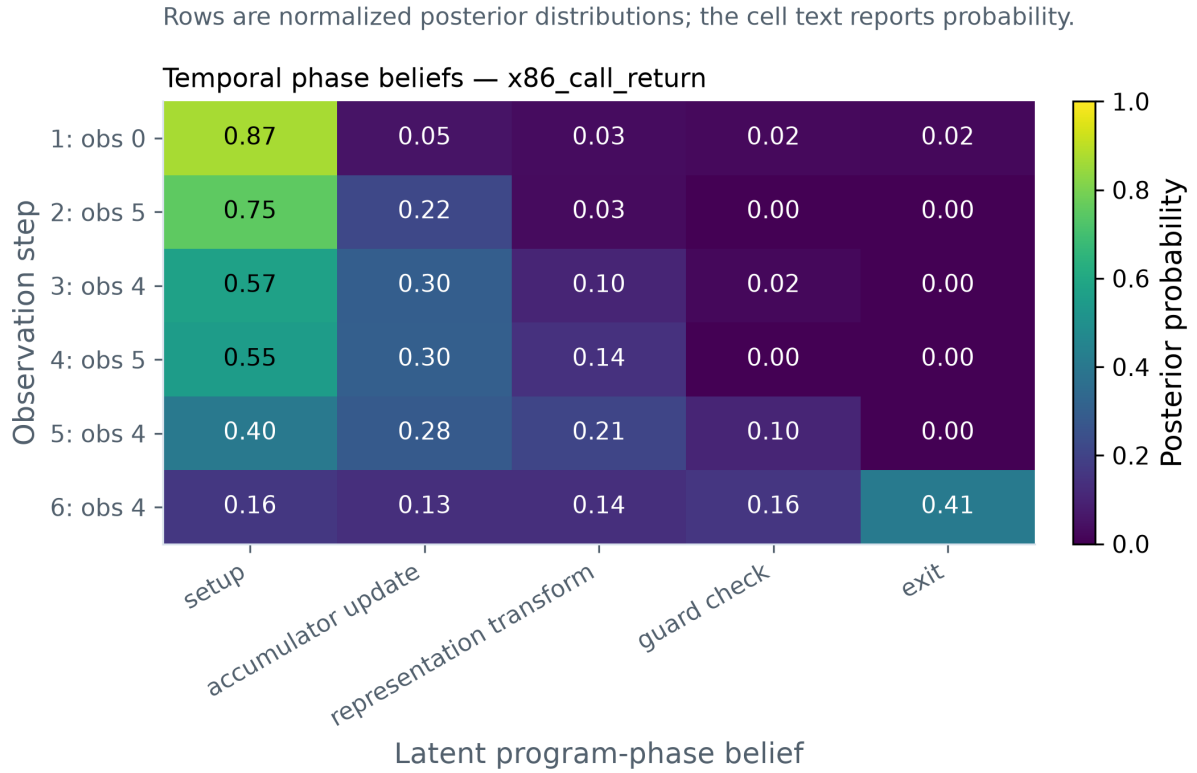


Figure 19: Temporal forward-filter beliefs for the x86_call_return fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation; the heatmap is a compact view of filtering uncertainty, not a discovered control-flow graph. The fixture runs a real call/ret stack roundtrip inside the bounded harness: the call pushes an in-span return address, the ret pad pops it back so control returns to the pop that reloads saved state, and the final ret is the exit. The call and ret instructions carry the control-transfer latent phase, so the guard_check column responds on steps that the surface observation vocabulary alone would not explain. The population is one transparent fixture and its six decoded instructions; phase labels are demonstrator evidence, so the figure supports temporal-filtering diagnostics only and makes no learned-semantics claim.

Rows are normalized posterior distributions; the cell text reports probability.

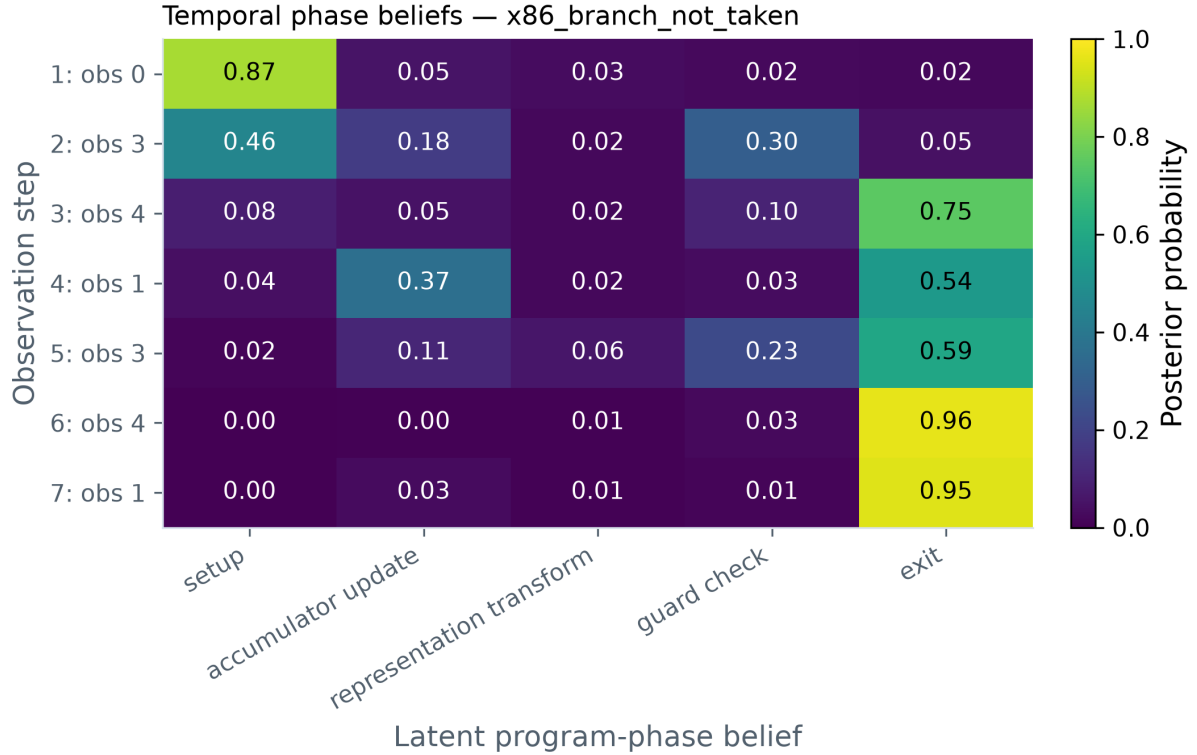


Figure 20: Temporal forward-filter beliefs for the x86_branch_not_taken fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation. Both guard phases in this fixture fall through: the accumulator value is neither greater than the first bound nor less than the second, so the `jg` and `jl` conditional branches are not taken and every statically decoded instruction executes. The heatmap shows the filter holding the `guard_check` phase across two consecutive comparison steps on each side of the accumulator update, making guarded-but-untaken control flow visible as belief mass rather than as an executed jump. The population is one transparent fixture and its seven decoded instructions; phase labels are demonstrator evidence, so the figure supports temporal-filtering diagnostics only and makes no learned-semantics claim.

Rows are normalized posterior distributions; the cell text reports probability.

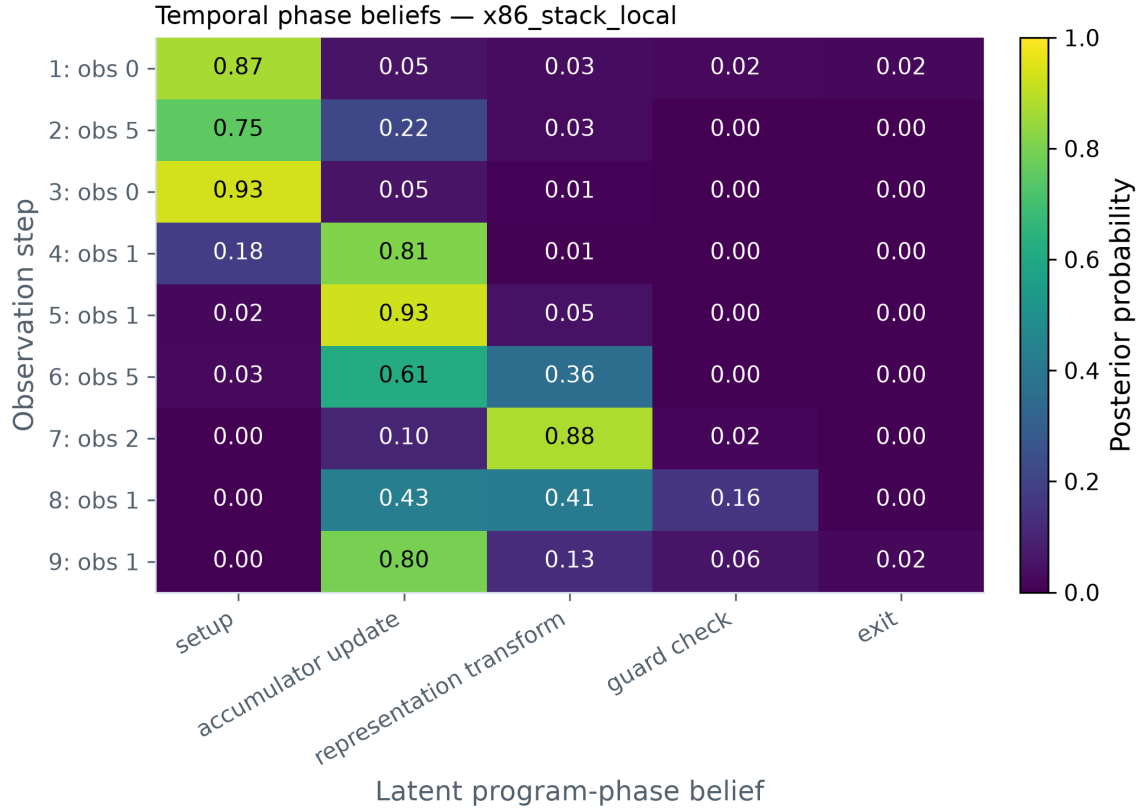


Figure 21: Temporal forward-filter beliefs for the x86_stack_local fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation; observation IDs are row labels and are not numeric magnitudes. The fixture keeps bounded stack-local state: the push writes a scratch slot below the stack pointer and the pop reads it back into a non-accumulator register, so accumulator arithmetic interleaves with real stack-memory effects. The setup and accumulator_update phases each recur in separate contiguous runs, which makes the temporal model’s state persistence observable as repeated belief columns rather than a single sweep. The population is one transparent fixture and its nine decoded instructions; phase labels are demonstrator evidence, so the figure supports temporal-filtering diagnostics only and makes no learned-semantics claim.

Rows are normalized posterior distributions; the cell text reports probability.

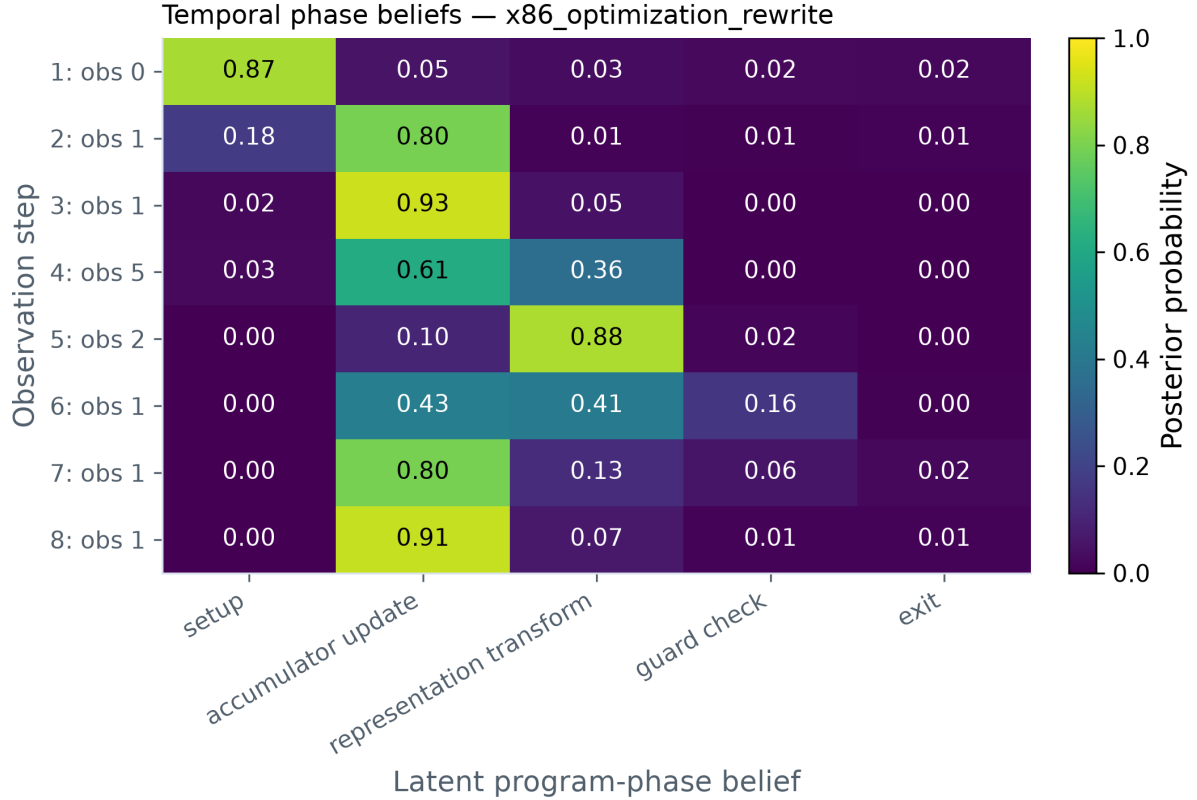


Figure 22: Temporal forward-filter beliefs for the x86_optimization_rewrite fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation. The fixture holds the classic optimization-sensitive motif pair: the add-with-immediate and inc instructions share one arithmetic observation category while differing in flag effects, and the lea rewrite lands in the residual other category even though its latent phase is accumulator_update – exactly the ambiguity-as-data regime the corpus is designed to expose. The heatmap therefore shows belief mass splitting between the accumulator_update column and the residual category on the lea step. The population is one transparent fixture and its eight decoded instructions; phase labels are demonstrator evidence, so the figure supports temporal-filtering diagnostics only and makes no learned-semantics claim.

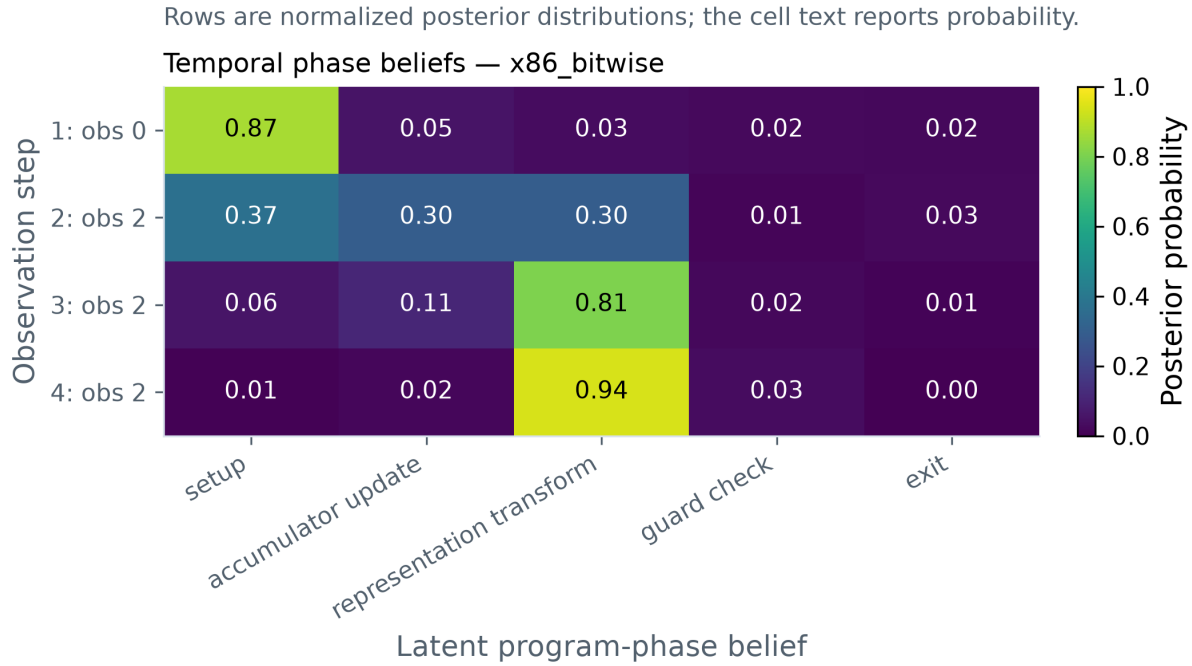


Figure 23: Temporal forward-filter beliefs for the x86_bitwise fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation; observation IDs are row labels and are not numeric magnitudes.

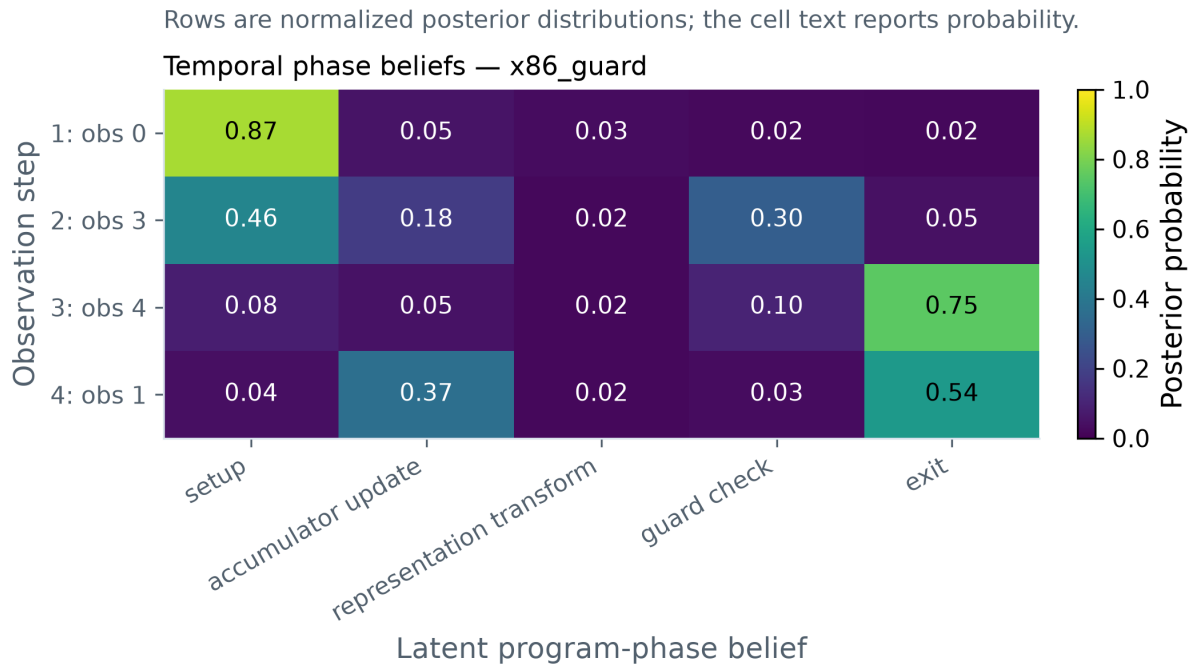


Figure 24: Temporal forward-filter beliefs for the x86_guard fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation; the heatmap is a compact view of filtering uncertainty, not a discovered control-flow graph.

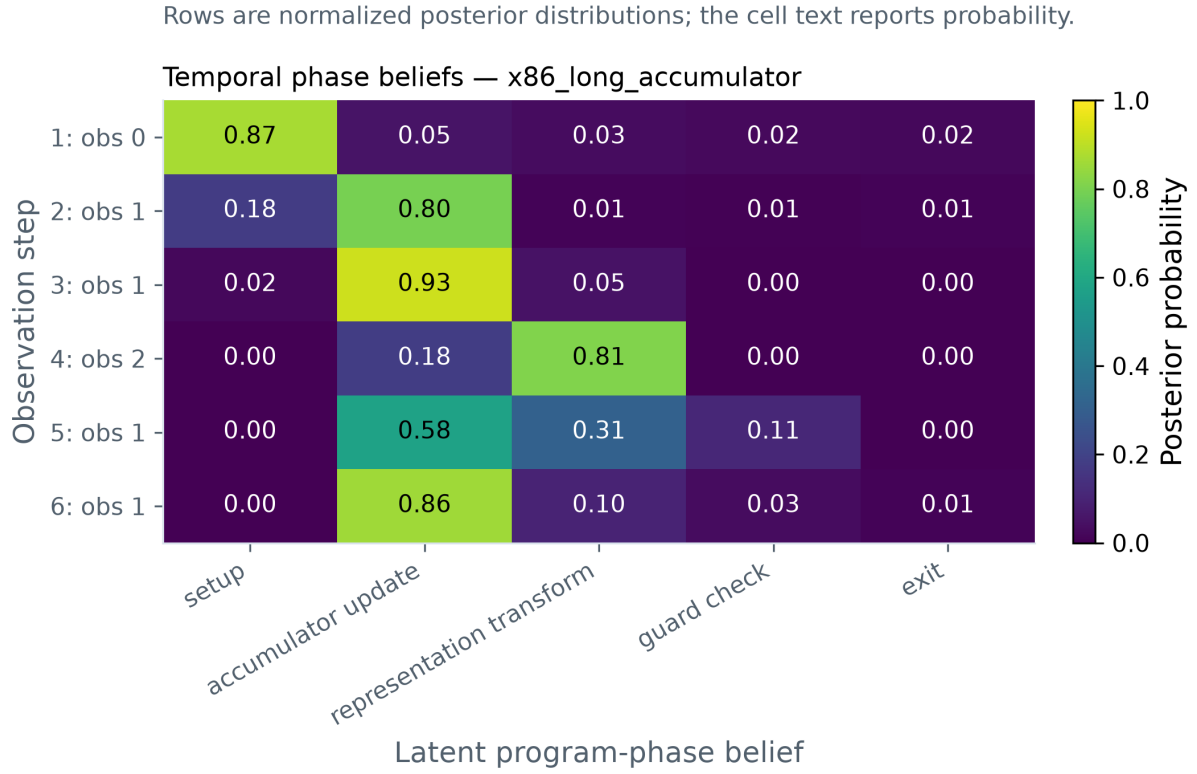


Figure 25: Temporal forward-filter beliefs for the x86_long_accumulator fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation; the longer fixture makes state persistence visible without changing the inference contract.

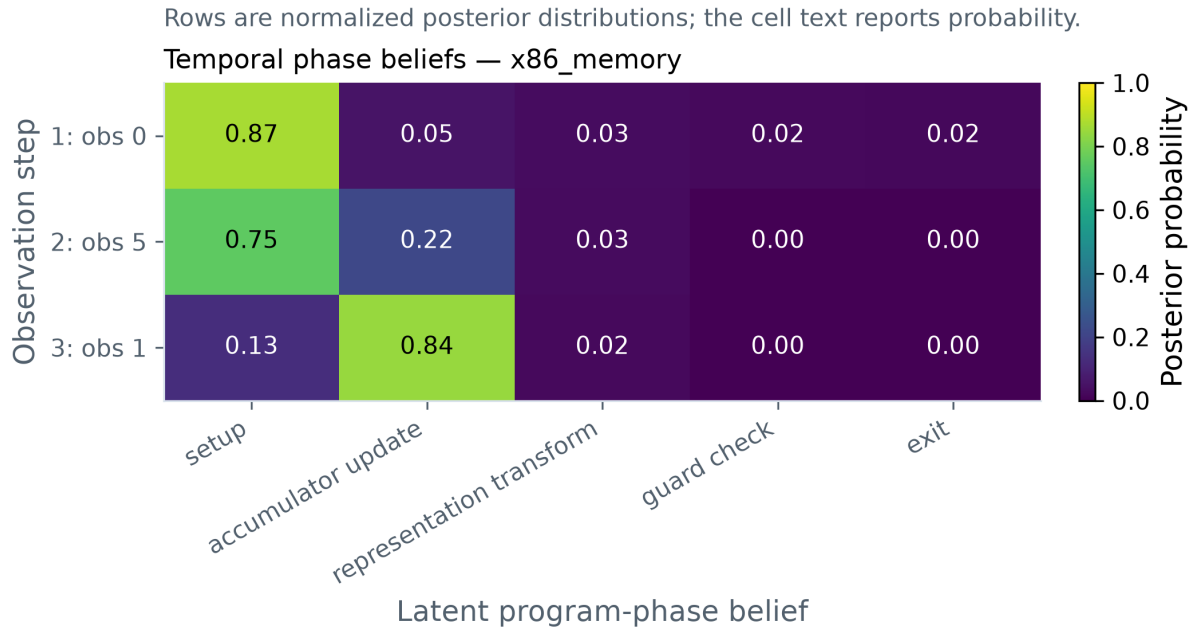


Figure 26: Temporal forward-filter beliefs for the x86_memory fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation; the fixture adds bounded stack-memory operands to the typed evidence surface without claiming memory semantics.

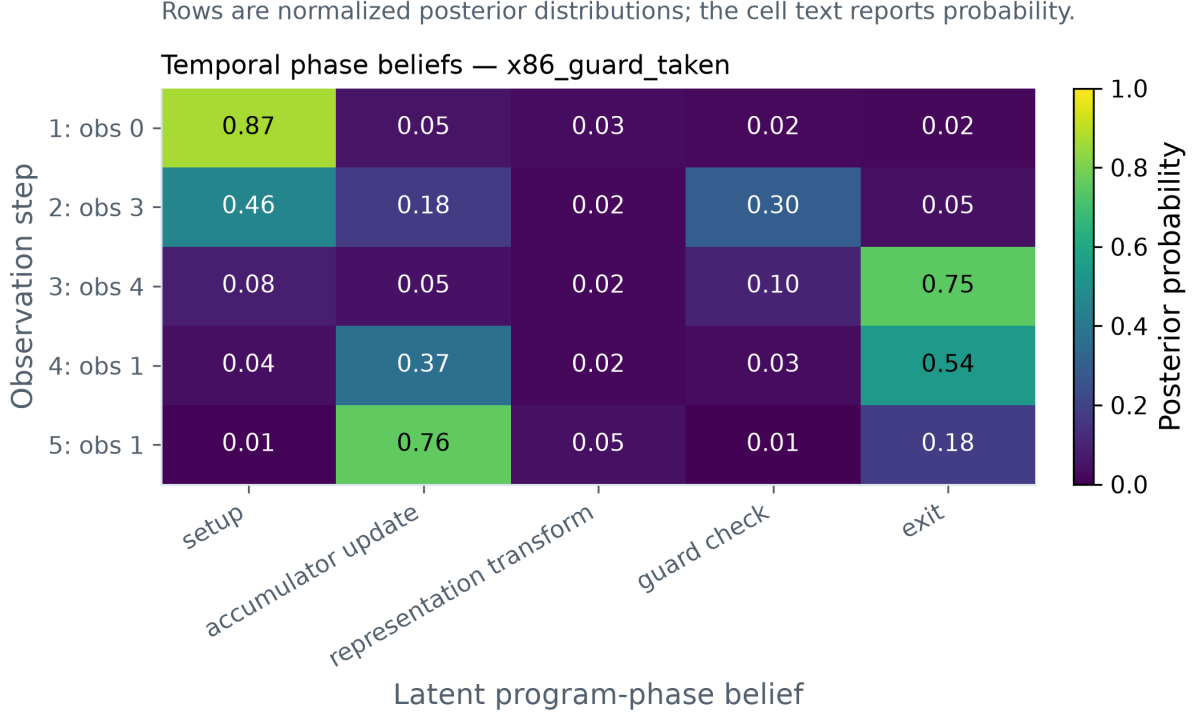


Figure 27: Temporal forward-filter beliefs for the x86_guard_taken fixture. Each row is a normalized posterior over 5 hand-authored phases after one decoded observation; the taken branch leaves one statically decoded instruction outside the bounded dynamic path.

and the action’s expected epistemic value is the mutual-information term

$$\text{EIG}(a \mid b_t) = \sum_{v \in V_f} P(v \mid b_t, a) \left[H(b_t) - H(b_{t+1}^{(v)}) \right]. \quad (26)$$

The demonstrator uses a common normalized action cost $c(a)=1$ and ranks actions by

$$G_{\text{sense}}(a \mid b_t) = -\text{EIG}(a \mid b_t) + \lambda c(a), \quad \lambda = 0.050. \quad (27)$$

Here `lambda` is a fixed configured weight, not a fitted utility or a physical cost conversion; the generated artifact also records the active-minus-fixed action-count contrast as an outcome. After selection, the observed field value updates the belief with the corresponding factor likelihood. The evaluation compares this predictive policy with source-order fixed sensing, a seeded-random permutation, and an explicitly inadmissible oracle that is allowed to inspect the realized outcome before choosing. All policies share the same fixture-level holdout, threshold, and action budget. This separation makes the no-peeking boundary testable and prevents a favorable realized outcome from being used to retroactively justify the active choice.

The action-conditioned output is a bounded sensor-choice mechanism experiment, not a claim about autonomous binary analysis. Costs are normalized action units rather than wall-clock measurements, and field values remain syntactic Capstone/Unicorn evidence rather than discovered program semantics.

3.3.2 Exact finite-horizon information-per-cost analysis

The action-conditioned experiment is paired with a smaller mechanism analysis whose optimization problem is exact within its declared finite support. The label-blind factorized model supplies one categorical sensor for each selected typed field. For sensor j with outcomes v and state-conditioned channel $L_j(v \mid s)$, belief b yields predictive probability

$$p_j(v | b) = \sum_s L_j(v | s) b(s), \quad b^{(v)}(s) = \frac{L_j(v | s) b(s)}{p_j(v | b)}. \quad (28)$$

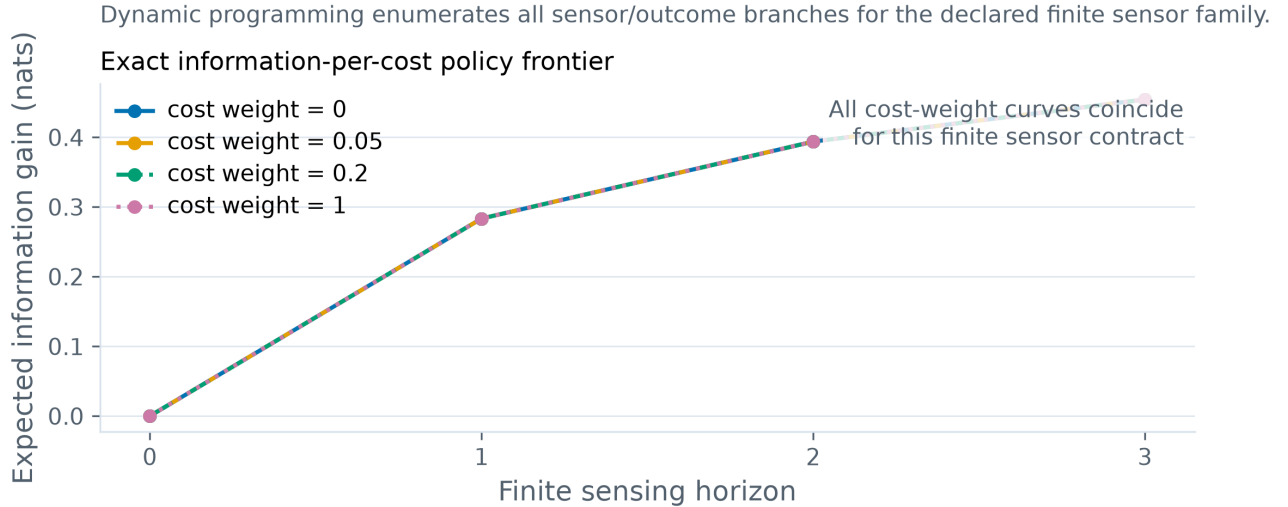
Let $\mathcal{J}_{-j}$ denote the remaining sensors in $\mathcal{J}$ after querying and removing j . The action objective Q_h and finite-horizon value J_h retain that remaining-set argument explicitly:

$$Q_h(b, j; \mathcal{J}) = \lambda c_j + \sum_{v: p_j(v|b) > 0} p_j(v | b) J_{h-1}(b^{(v)}; \mathcal{J}_{-j}),$$

$$J_h(b; \mathcal{J}) = \min_{j \in \mathcal{J}} Q_h(b, j; \mathcal{J}). \quad (29)$$

The terminal values are $J_0(b; \mathcal{J}) = H(b)$ and $J_h(b; \emptyset) = H(b)$, with declared sensor costs $c_j > 0$. Posterior updates are defined only for positive-probability outcomes. The implementation enumerates all positive-probability outcome branches and memoizes normalized beliefs, remaining sensors, horizon, and cost weight. It therefore supplies an exact result for the finite sensor family and horizons in the receipt, while making no claim about an infinite-horizon policy, continuous sensing, or operational acquisition time.

The source-bound artifact is `../output/data/active_information_frontier.json`, with 4 sensors and 16 frontier rows. Exact branch enumeration is recorded as `yes`; the frontier is visualized in fig. 28. That information frontier uses model-derived unit costs and is therefore a mechanism result, not a wall-clock law. The measured-cost receipt in the measured-sensing section below is a separate, deterministic/calibrated layer; neither artifact supports a universal cost claim or a policy guarantee outside the finite declared family.



Costs are model-derived proxies; this finite mechanism result is not a wall-clock law.

Figure 28: Exact finite-horizon information-per-cost analysis over the declared label-blind typed-field sensor family of 4 channels. Lines show expected information gain in nats as the sensing horizon grows under four cost weights; every point is obtained by exhaustive dynamic programming over sensor/outcome branches, with no Monte Carlo approximation. Curves may coincide when the finite contract selects the same information frontier at each weight. The plotted frontier uses unit action costs from the finite model channel; the separate sensor-cost receipt reports deterministic vectors and local timing calibration, so this panel is a mechanism frontier rather than a deployment cost law [Friston et al., 2017, Parr et al., 2022].

The certificate frontier (fig. 29) makes the cost model auditable at the same time as it makes the optimization boundary visible. Exact dynamic programming supplies the finite oracle and its regret/optimality gap; any timing overlay is a local calibration with its own machine receipt. The visual must therefore be read as two linked coordinate systems, not as a universal exchange rate between information and hardware time.

Finite sensing policy certificates and cost frontier

The oracle is an inadmissible reference for certification; deterministic counts and optional local wall-clock calibration remain separate quantities.

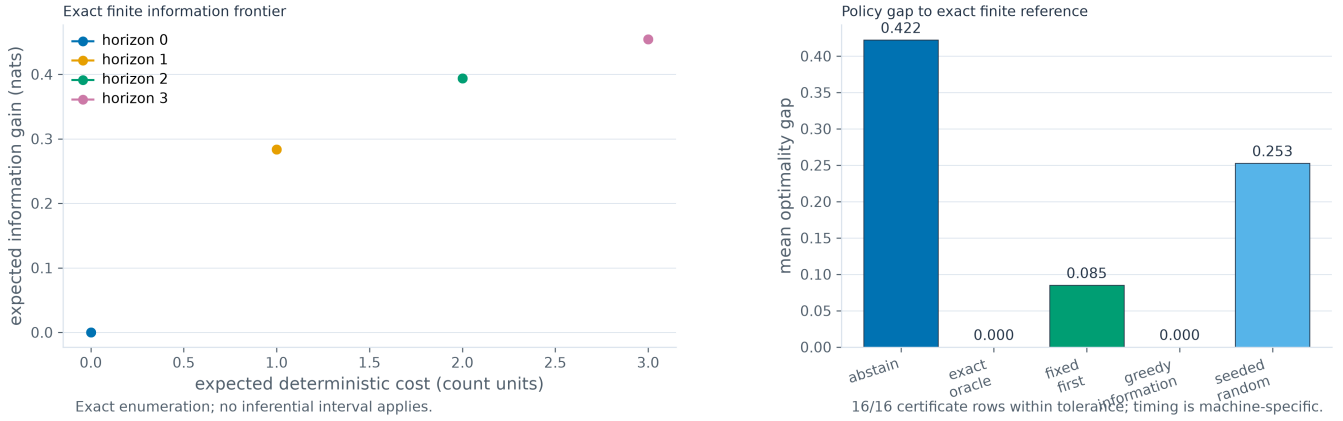


Figure 29: The policy frontier contains 16 exact dynamic-program rows across horizons 0, 1, 2, 3 and 16 finite certificate rows. The left panel plots expected information gain in nats against deterministic count cost; the right panel summarizes optimality gaps for implemented and comparison policies when present. Exact finite enumeration has no sampling confidence interval. Optional wall-clock values are local calibration measurements and remain separate from deterministic counts. The exact oracle is an inadmissible reference used for regret and gap calculations. Any tolerance certificate applies only to the frozen sensor family, state space, and horizon, not to arbitrary binaries, hardware, or a universal sensing law.

A labelled corpus of 10 transparent byte fixtures, each carrying a hand-authored ground-truth phase sequence aligned to its decoded instructions, supports the project’s first quantitative evaluation: decoding accuracy, calibration, a free-energy convergence summary, and an identity-transition, temporal, and Viterbi comparison, each guarded by shuffled-label and uniform-belief negative controls so that a reported gain cannot be vacuous [Heins et al., 2022].

The corpus is a deliberately small, hand-labelled demonstrator rather than a sample of arbitrary binaries. The confidence intervals reported below are descriptive Wilson score intervals around pooled per-step hit counts; steps within a fixture are dependent. In addition to the transparent hand-authored model, the project now reports a leakage-safe supervised fit: each leave-one-fixture-out row estimates A, B, and D on the other fixtures only, then scores the excluded fixture. The calibration summaries follow the reliability-diagram and ECE framing of [Guo et al., 2017], while remaining fixture-specific. Wilson score intervals are used because the displayed quantities are binomial hit-rate summaries and the Wilson construction behaves more reliably than a Wald approximation for small denominators [Brown et al., 2001]. This choice does not remove the dependence among steps or broaden the corpus into a population sample.

The training/development/held-out assignment is a declared fixture inventory. The temporal estimation reports below use leave-one-fixture-out folds: each fold excludes its scored fixture but may train on fixtures assigned to the inventory’s other partitions. They do not report a model trained once on the fixed training partition and evaluated once on its held-out partition. Related authored fixtures are also not independent source lineages. The fold results therefore support a complete-fixture exclusion diagnostic, with no claim of lineage-disjoint generalization.

The corpus expands the original six authored programs with four designed fixtures: an in-span call/**ret** roundtrip (the call pushes a return address inside the decoded span and the **ret** pads it back), a not-taken guard pair (two conditional branches whose guards both fall through, so every statically decoded instruction executes), a stack-local fixture (a push/pop pair keeps bounded scratch state below the stack pointer while accumulator arithmetic continues), and an optimization-rewrite fixture (the add/immediate, **inc**, and **lea** rewrite motif in which **add** and **inc** share one arithmetic observation category and the **lea** rewrite lands in the residual **other** category despite its accumulator-update latent phase). The temporal family split (4/2/4 training/development/held_out) was frozen in the protocol before any scoring under protocol v2.1, with each fixture assigned wholesale to exactly one partition and no family overlap.

The expanded experiments are governed by one versioned protocol rather than by independently edited scripts. The protocol receipt is `../output/data/research_protocol.json`, identified as `active-assembly-open-roadmap-v2-2026-10-13` and hashed as `d394c9264fb64cb177e42910745deb84e9f00fca55d04159653b6ab89e836b7b`. Every new

scientific receipt records that hash, its own schema version, a producer-source digest, input hashes or source manifests, and the available toolchain versions. A generated number therefore has two obligations: it must be reproducible from its producer, and it must remain within the frozen scientific contract. A byte-for-byte replay of an unfrozen protocol would only reproduce an implementation accident.

The protocol freezes the observational state, interventions, missing/unseen tokens, supported instruction boundary, family-level partitions, model families, seeds, primary metrics, clustered uncertainty, multiplicity policy, deterministic sensing costs, and the external-source safety boundary. The public input directory in the local replication bundle is label-blind. A missing field is encoded as `__MISSING__`; a value absent from training support is encoded as `__UNKNOWN__`. Neither is replaced by zero, an empty string, or a silently dropped row. Unsupported instructions, unavailable compilers, build failures, and underpowered strata remain represented as structured statuses.

3.3.3 Protocol-v2 artifact envelope and manuscript closure

The protocol contract is layered rather than destructive. Every project-owned JSON artifact currently carries the common receipt schema 2.0 above its domain-specific `schema_version`. The common envelope requires 10 fields: the protocol identifier and digest, producer revision, input hashes, toolchain versions, evidence status, limitations, unsupported cases, and underpowered cases. This separation keeps a trace-schema migration distinguishable from a provenance-contract migration and allows domain validators to retain their existing interpretation.

The current generated surface contains 51 JSON artifacts, of which 0 fail the receipt audit. Producers write through an atomic temporary-file replacement, so a failed serialization cannot expose a partially written receipt. Legacy payloads may be migrated only by an explicit producer-stage command; validation is read-only and fails closed when a payload bypasses that stage. The canonical audit command is `uv run python scripts/run_scientific_program.py audit-artifacts`. This is a provenance and publication- integrity mechanism informed by PROV-O, RO-Crate, and reproducible-computing practice [Lebo et al., 2013, Sefton et al., 2023, Sandve et al., 2013]. Its receipt count is not a scientific sample size, and its passing state cannot promote semantic identifiability, external validity, or independent replication. The local versus independently reproduced distinction follows the terminology used by the ACM artifact-review policy [Association for Computing Machinery, 2020].

3.4 Assembly state, normalized intermediate representation, and flags

The machine-facing formalism is a normalized instruction intermediate representation (IR), not a claim that a mnemonic string is a semantic object. For a decoded instruction i , the IR records

$$I_i = (\text{pc}_i, \text{op}_i, \text{dst}_i, \text{src}_i, \text{imm}_i, \text{mem}_i, \text{reads}_i, \text{writes}_i, \text{flags}_i^r, \text{flags}_i^w, \text{target}_i), \quad (30)$$

where `pc` is the instruction address, `op` is a normalized operation family, `dst/src` are canonical register or memory terms, and the remaining fields preserve explicit register, flag, memory, and control-flow effects. The canonicalization is conservative: it can identify a bounded class of equivalent operand descriptions, but it does not infer calling conventions, alias sets, compiler intent, or source-level names.

The common bounded state is

$$\sigma = (pc, G, F, M, K, i, x), \quad (31)$$

with program counter pc , modeled general-purpose registers G , condition flags F , bounded byte-addressable memory M , bounded stack slice K , decoded IR record i , and an explicit exceptional/unsupported marker x . The modeled flags are those read or written by the supported operation; they are not treated as globally known architectural state when an instruction does not define them. A supported small-step transition has the form

Definition 4 (Normalized bounded machine state). The register component is the explicit 16-register file $G = (g_0, \dots, g_{15}) \in (\mathbb{W}_{64})^{16}$, with RIP represented by pc rather than hidden in memory. A byte-addressable memory is a finite partial map $M : \mathbb{W}_{64} \rightharpoonup \mathbb{W}_8$ whose declared domain is the bounded memory interval; the stack slice K is a named subinterval of that domain. Each flag is a pair `(value, definedness)`, so a preserved or undefined flag is never coerced to a Boolean observation. The state also records the instruction index, the normalized instruction, and one of `running`, `halted`, `unsupported`, `exceptional`, or `bound_exceeded`.

Operand width is part of the transition, not an annotation added after the fact. For a value v written to a register whose current value is g , the write operator is

$$\text{write}_w(g, v) = \begin{cases} v \bmod 2^{64}, & w = 64, \\ v \bmod 2^{32}, & w = 32, \\ (g \& \neg(2^w - 1)) \mid (v \bmod 2^w), & w \in \{8, 16\}. \end{cases} \quad (32)$$

Thus a 32-bit write zero-extends into the 64-bit architectural register, while 8- and 16-bit writes preserve their untouched upper bits. For a memory operand with base b , index j , scale $s \in \{1, 2, 4, 8\}$, and signed displacement d , the effective address is

$$a = (b + s j + d) \bmod 2^{64}, \quad \text{valid}(a, n) \iff [a, a + n - 1] \subseteq \text{dom}(M). \quad (33)$$

Loads and stores are admitted only when `valid` holds for every byte. The reference interpreter does not turn a failed bounds check into a wrapped or zero-filled value; it returns a structured bound or unsupported result.

For w -bit operands a, b and result r , the arithmetic flags are defined by

$$\begin{aligned} CF_+ &= \mathbf{1}[a + b \geq 2^w], & CF_- &= \mathbf{1}[a < b], \\ ZF &= \mathbf{1}[r = 0], & SF &= \text{bit}_{w-1}(r), \\ OF_+ &= \mathbf{1}[\text{sign}(a) = \text{sign}(b) \neq \text{sign}(r)], & & \\ OF_- &= \mathbf{1}[\text{sign}(a) \neq \text{sign}(b) \wedge \text{sign}(r) \neq \text{sign}(a)]. & & \end{aligned} \quad (34)$$

ADC and SBB include the incoming defined CF in the arithmetic before computing their outgoing flags. INC and DEC use the corresponding add/subtract result but preserve the incoming CF; CMP uses subtraction to set flags without writing its destination, and TEST uses a logical AND without writing either operand. Logical and shift operations mark only the flags explicitly defined by the contract. These cases are separately exercised at zero, all-ones, sign-transition, carry/borrow, and shift-boundary values.

Proposition 3 (Determinism of a supported bounded step). For a normalized instruction whose operands, defined flag reads, memory range, and control-flow target satisfy the protocol bounds, the reference transition $\langle \sigma, I_i \rangle \rightarrow \sigma'$ is deterministic. If any precondition is not satisfied, the transition returns one structured non-running status rather than an architectural value. This proposition is about the implemented finite interpreter; it is not a completeness or total-correctness theorem for x86-64.

$$\langle \sigma, I_i \rangle \longrightarrow \sigma' \quad (35)$$

and is defined only when operand widths, address bounds, flag dependencies, and exceptional behavior are within the declared contract. For example, a 64-bit addition updates the destination modulo 2^{64} and computes its declared carry, zero, sign, and overflow bits; a compare updates flags without writing the destination; and a conditional branch reads only the flags listed by its condition. `movzx`, `movsx`, `lea`, bounded loads/stores, stack operations, calls, returns, and branches are admitted only when their width and target behavior fit the bounded state. SIMD, floating point, privileged/system state, self-modifying code, unmodeled exceptions, undefined behavior, and unbounded aliasing are rejected with an unsupported reason.

The assembly listing in [lst. 1](#) remains the human-readable projection of this IR. The new correspondence receipt compares the pre-state, decoded address, post-state, modeled flags, memory effects, and control-flow target at every bounded step; terminal equality alone is not accepted as a trace proof. The Intel manuals specify architectural behavior, Capstone supplies a decoding/API boundary, Unicorn supplies bounded execution, and the Lean fragment supplies independently checked algebra for a much smaller closed subset [[Intel Corporation, 2026](#), [Capstone Engine Project, 2026](#), [Unicorn Engine Project, 2026](#), [Lean Community, 2026](#)].

Listing 2 Illustrative normalized instruction records; the generated receipt retains the full source-bound correspondence and unsupported boundary.

```
0x1000 MOV RAX <- 1
  flags: preserved
  memory/target: none
0x1005 ADD RAX <- RAX + 2
  flags written: CF, PF, AF, ZF, SF, OF
  memory/target: none
0x1009 JCC if ZF: target
  flags read: ZF; flags preserved
  memory/target: bounded internal target
```

3.5 Formal correspondence and transfer receipts

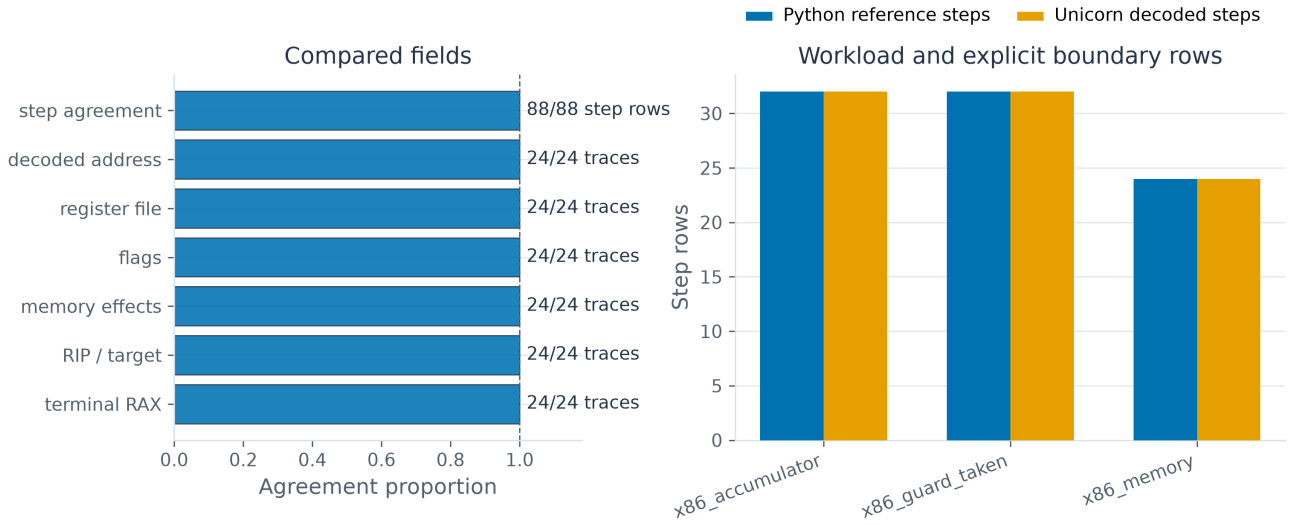
The Python reference interpreter is executed on transparent repository-owned fixtures and compared with real Capstone decoding and bounded Unicorn traces. The receipt `../output/data/formal_trace_correspondence.json` covers 3 fixtures under 8 declared interventions each, with 88 reference step rows. Its status is `validated_bounded`, and the comparison is visualized in fig. 30. The comparison is all-step and fail-closed: an unsupported row, environment error, or mismatch cannot be converted into a terminal-only success. The Lean prerequisite is reported separately in `../output/data/formal_verification.json`; a generated Lean bridge now serializes one shared bounded state with all 16 general-purpose registers, RIP, modeled flag definedness, memory and stack bounds, step index, and running status. The bridge covers 6 transparent fixtures with 20 generated machine-state theorems, all kernel-checked (`true`) under the `leanprover/lean4:v4.35.0` toolchain, alongside 71 formal theorems with an axiom report that is clean (`true`). The kernel-checked fixture invariants cover the 64-bit RAX increment, preservation of the other registers and carry flag, step advancement, and bound preservation. A compiled Lean theorem is not silently promoted to a proof of the Python lifter, the Capstone/Unicorn path, or the complete x86-64 ISA.

The transfer track is deliberately separate from the transparent-execution track. Portable source families are compiled with the exact locally available Clang target command, object and disassembly hashes are retained, and emitted objects are statically decoded without execution. The Clang manual defines the compiler-driver and target-selection interface used for this acquisition [LLVM Project, 2026a]. Optional NASM and Rust cells use their documented object-format and static-emission interfaces [NASM Project, 2026, The Rust Project Developers, 2026]; a missing compiler or target remains an unavailable cell. The design is provenance-oriented rather than a compiler-correctness claim; translation validation is the relevant precedent for treating each produced translation as an individually checkable artifact [Necula, 2000]. The current receipt is `../output/data/compiler_transfer.json`: a 184-cell matrix over the compilers `clang`, `nasm`, `rustc`, the object formats `elf`, `macho`, and the architectures `aarch64`, `aarch64_linux`, `x86_64`, `x86_64_linux`, of which 128 cells are validated static objects and 56 are explicit unavailable rows retaining their compiler diagnostics. The separate licensed-source receipt is `underpowered`; it therefore cannot support an application-facing effectiveness claim. A target absent from the local compiler is a missing capability, not evidence about that architecture. This is the correct interpretation of the compiler capability summary in fig. 31.

The separate licensed-source receipt is `../output/data/external_source_validation.json` with status `underpowered`, 5 manifested projects, and 100 labelable examples. The pinned manifest records 10 source rows spanning 5 projects and two revisions per project. Static coverage and source-bound labels are separate evidence surfaces. The static receipt `../output/data/external_static_validation.json` is `validated`: it retains 10 successfully compiled-and-decoded rows of 10; no static object is treated as an equivalence label. The dedicated label receipt `../output/data/external_label_contract.json` is `validated` and contains 100 accepted rows: 40 lexical-equivalence witnesses, 40 contract-directed one-change near misses, and 20 identity negative controls, with 0 explicit exclusions. The `libyaml` project is reserved for the held-out split. A trailing comment is a preprocessing-token witness, not a universal semantic-equivalence theorem; near-miss rows are source-contract controls and are not executed behavioral labels. Every accepted row binds an immutable revision, license-file digest, archive digest, compiler argument vector, target triple, source edit and argument, base/pair source bytes, object hashes, canonical disassembly hashes, and Capstone coverage; emitted objects are not executed by the canonical pipeline.

The external-source gate is therefore a safety and provenance interface, not a permission to execute downloaded software. Acquisition requires the explicit network opt-in command and a pinned manifest; replay, scoring, and manuscript generation remain offline. The current receipt records 12 gate fields and keeps failed static rows, source-contract exclusions, and the missing separate replay as `underpowered` or `not_yet_observed` evidence rather than

Python reference ↔ Capstone ↔ bounded Unicorn correspondence



24 intervention traces; 88 reference step rows. Boundary: 0 unsupported traces; 0 bound/exception/unsupported steps. No inferential interval.

Agreement is reported as numerator/denominator at the declared field grain; the receipt is bounded, not complete x86-64 semantics.

Figure 30: Field-level correspondence receipt for the declared transparent x86-64 subset. The left panel uses numerator/denominator marks for step agreement, decoded addresses, register state, modeled flags, memory effects, control targets, and terminal RAX; the right panel reports Python-reference and Unicorn step workloads by fixture. The receipt covers 3 fixtures and 8 bounded interventions per fixture, with explicit unsupported and bound/exception rows. These are exact fixture-level checks, so inferential intervals are not applicable. A generated Word64 boundary fixture is also kernel-checked in Lean. The result is a bounded correspondence instrument, not a complete proof of x86-64 semantics.

inventing a denominator. The local compiler capability receipt is `../output/data/external_capabilities.json` with status `validated` and 4 target probes that produced a static object. A compiler-advertised target is not counted unless this empirical probe succeeds.

The acquisition attempt itself is separately recorded in `../output/data/external_acquisition.json` with status `validated`, 10 retained source rows, and recorded network use `no`. That receipt binds the manifest and producer hashes, preserves one row per pinned source, and distinguishes a verified local-cache reuse from a network download. Downloads are published only after archive-digest verification; extraction is staged, archive links are rejected, and the declared license bytes are checked before publication. The validator repeats those checks without network access. This is a freshness and safety improvement for the source-bound corpus, not evidence that the external build was independently replayed or that any source-level contract is a universal behavioral- equivalence label.

Compiler capability and static decoding

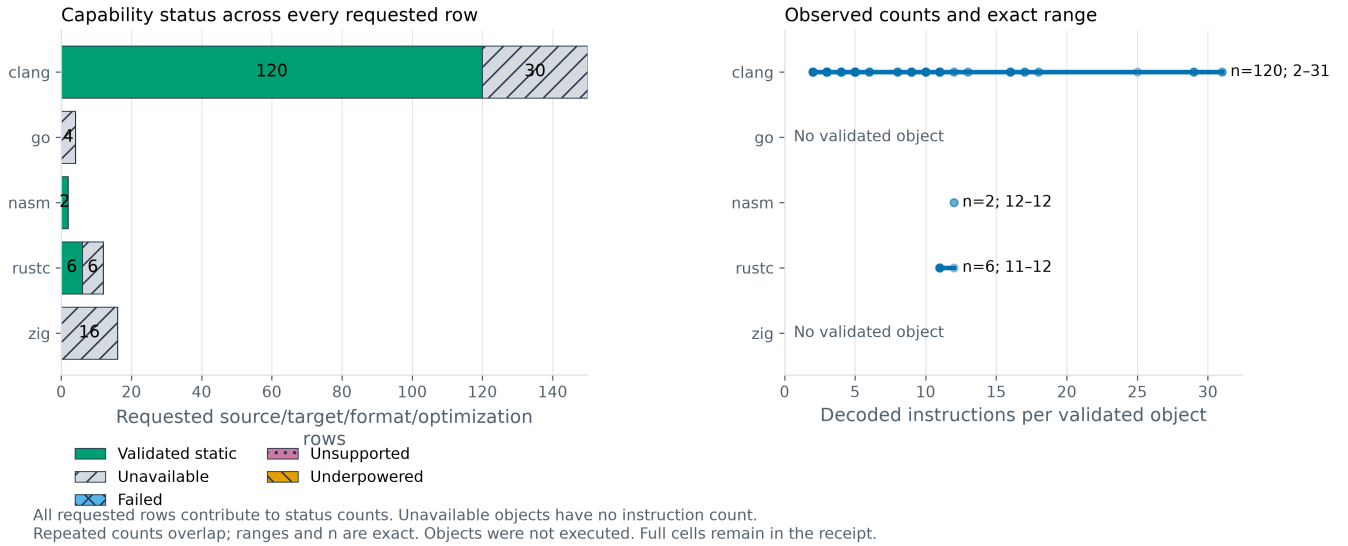


Figure 31: Compiler/source/architecture capability receipt for 128 locally compiled and statically disassembled objects across the recorded multi-toolchain matrix of compilers, target triples, Mach-O and ELF object formats, and per-compiler optimization vocabularies. Stacked bars count every requested source/target/format/optimization row by compiler and explicit capability status; text and hatching retain unavailable, failed, unsupported and underpowered outcomes. The receipt includes 4 validated architecture target(s) and 56 unavailable row(s); the right panel shows decoded instruction counts for all validated objects, with each compiler’s exact range and denominator. Repeated points overlap; unavailable objects have no instruction-count point. Full cells and commands remain in the receipt. Counts are deterministic provenance diagnostics with no population uncertainty. No emitted object was executed, so the visual supports local capability and provenance claims only, not compiler correctness, behavioral transfer, or architecture-general performance.

The hierarchy (fig. 32) is the reason the analysis reports family-level uncertainty. Transformations, interventions, traces, and instruction steps are useful observations, but they share source lineage and cannot be treated as a flat independent sample. Fixing the family split before fitting also prevents the development and test boundaries from being reconstructed from row-level features or source order.

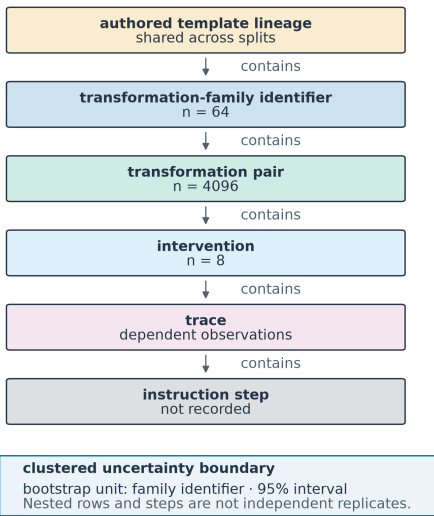
3.6 Quotient classes and co-primary classifier portfolio

The quotient endpoint is pairwise by design. A pair is labelled positive when the two transparent programs have equal terminal `RAX` for every intervention in the frozen catalog; it is negative when at least one intervention separates the terminal outputs. The relation is generated before model fitting and is not a mnemonic label, family identifier, transformation name, or model output. This is a bounded observational quotient, not complete program equivalence. In formal-methods terms it is an observation relation over a finite intervention catalog, not a bisimulation relation preserved by every transition [Sangiorgi, 2011].

Statistical hierarchy and leakage control

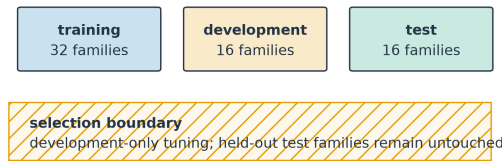
Family identifiers define resampling and fitting boundaries; shared authored lineages prevent a population-independence interpretation.

Nested dependence hierarchy



Fixed family partition

The split is declared before fitting; only development identifiers control selection and stopping.



Reading rule

Identifiers are disjoint; authored lineages are shared.
More rows do not create independent sources or population coverage.

split contract: family-disjoint; development-only selection

Figure 32: The statistical hierarchy contains 64 transformation-family identifiers, 4096 pair rows, and 8 classifier interventions in the current receipt. Family identifiers group transformation pairs; pairs are evaluated under interventions, producing traces with dependent instruction steps. Identifiers are disjoint across splits, but authored template lineages are shared. The resampling unit is the held-out family identifier, not an independently sampled source lineage. The declared cluster-bootstrap contract uses a 0.95 confidence level. Training, development, and test families are fixed before fitting, with selection and stopping restricted to development data. The visual explains the dependence structure and interval meaning; it does not license instruction-step pseudoreplication or turn row counts into independent evidence.

The classifier receipt is `../output/data/quotient_classifier.json` with status `validated_bounded`; its held-out portfolio comparison is visualized in fig. 33. It uses 64 families and 4096 pair rows, split into 32 training, 16 development, and 16 held-out families. No family occurs in more than one partition. The calibrated interpretable model is a regularized binary logistic classifier over typed trace summaries; the temporal model is a class-conditional first-order Markov likelihood over typed token sequences; and the recurrent model is a fixed one-layer NumPy tanh recurrent network. The three are co-primary. A typed-trace summary is retained as the historical surface comparator, and six additional raw surface baselines are fit independently from training-only vocabularies: opcode histograms, hashed byte bigrams, mnemonic counts, instruction-length summaries, control-flow summaries, and memory-event counts. These baselines are not collapsed into one favorable “surface” score; each receives the same held-out predictions, family-level bootstrap, and development/test separation as the model portfolio.

The fixed iteration budgets and stopping rules are part of the protocol, as is the eight-state intervention catalog (8 initial register values): the logistic and recurrent fits do not inspect the held-out test set, while the Markov fit is a smoothed closed-form count estimator. Five deterministic seeds are rerun for each stochastic family and their range is recorded separately from the source-family bootstrap intervals. The recurrent implementation also emits a finite-difference gradient receipt; the checked derivative error is a software correctness diagnostic, not a performance interval.

For primary metric m and model k , balanced accuracy is the mean of positive and negative recall, while macro-F1 averages the F1 values for both quotient classes. Let u index held-out transformation-family identifiers and let $\hat{\theta}_{k,m}^{(b)}$ be the metric after resampling whole families with replacement. Resampling the family identifier, rather than treating repeated intervention rows as independent, is the declared dependence safeguard; the small-cluster caveat remains explicit [Cameron et al., 2008]. These named transformation families share authored template lineages; disjoint identifiers do not establish independent source sampling. The intervals and prediction swaps are conditional diagnostics on the finite inventory, with no population-generalization coverage claim. The reported interval is the 2.5th–97.5th percentile of the frozen 95% cluster bootstrap:

$$\text{CI}_{k,m} = [Q_{0.025}(\hat{\theta}_{k,m}^{(b)}), Q_{0.975}(\hat{\theta}_{k,m}^{(b)})]. \quad (36)$$

Brier score and fixed-bin expected calibration error are retained alongside the primary metrics. The five-percentage-point practical margin is retained. Protocol v2.2 separates percentile bootstrap intervals from significance testing: an uncentered bootstrap fraction below zero is a descriptive tail fraction, not a null p-value. The paired randomization diagnostic swaps all predictions in each held-out family between the model and baseline, recomputes both metrics, and uses inclusive ties with $(b+1)/(B+1)$ for B Monte Carlo draws [Phipson and Smyth, 2010]. Its null assumes prediction-block exchangeability within families, conditional on the fitted models. It does not include model-refitting uncertainty. This is a retrospective methods correction; superiority claims are withheld pending a prospectively frozen independent comparison. Holm adjustment [Holm, 1979] spans the co-primary model-by-baseline-by-metric comparison family. The current held-out values are 0.671 / 0.654 for the interpretable model, 0.775 / 0.772 for the temporal model, 0.485 / 0.340 for the recurrent model, and 0.674 / 0.657 for the surface baseline. These numbers are reported as a portfolio: overlapping clustered intervals, development/test differences, and the small number of declared families do not support a superiority claim. The retrospective correction withholds all such claims; a future prospectively frozen comparison must satisfy the practical margin and inference rule on independent evidence. The receipt retains 6 raw baseline models and 18 model-by-baseline comparison rows; no post-hoc baseline winner is promoted.

The construction diagram (fig. 34) separates three operations that are often conflated: declaring a source-level pair role, applying the finite intervention catalog, and assigning an observational label from the resulting normalized traces. Only the last operation supplies the classifier target. The relation is useful for a controlled benchmark because it is reproducible; it is not a claim that two programs agree under every input, architecture, exception, or calling convention.

3.6.1 Exact intervention power and finite separating sets

The quotient relation is only as informative as its declared intervention catalog. We therefore compute intervention power directly from the source-bound terminal-output matrix, before any classifier is fitted. For the separate representation-invariance pilot, let $\mathcal{U}_{\text{pilot}}$ be its four initial-state interventions and let $e_{p,u} \in \{0,1\}$ indicate whether pair p has equal terminal output under intervention u . For every non-empty subset $S \subseteq \mathcal{U}_{\text{pilot}}$, the near-miss separation and equivalent-pair consistency are

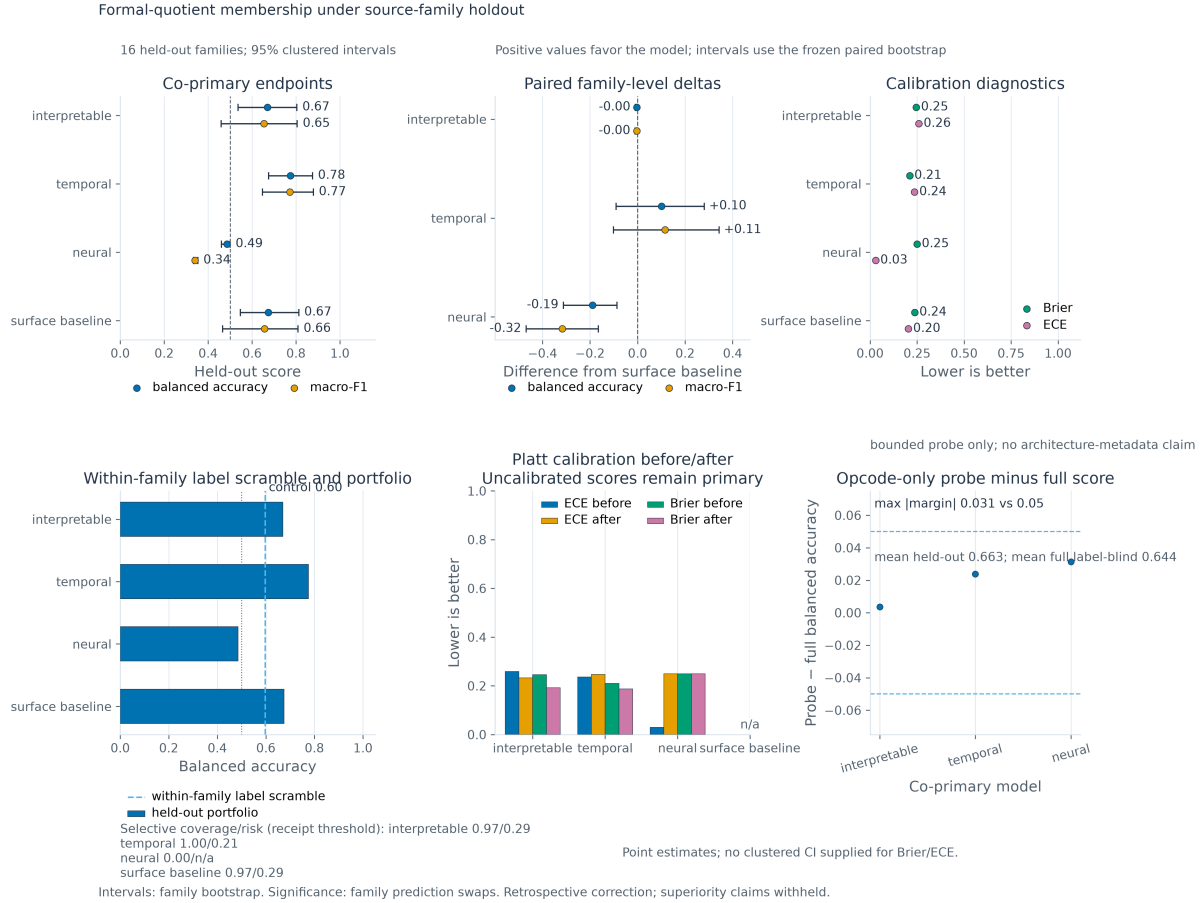
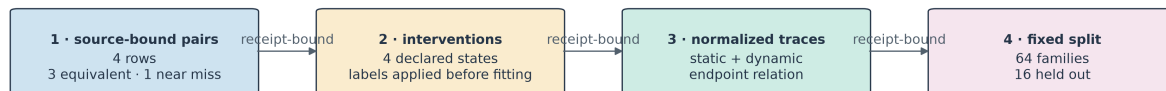


Figure 33: Co-primary binary classification of the declared finite quotient relation under source-family holdout. The left panel shows balanced accuracy and macro-F1 as model-specific dot-and-interval estimates; the second panel shows paired model-minus-surface-baseline deltas with source-family-clustered 95% bootstrap intervals; and the third panel shows Brier score and ten-bin expected calibration error as point diagnostics, with selective coverage and risk reported below. Receipt-supplied companion panels show the family-shuffle negative control beside the co-primary models, Platt-scaled calibration error and Brier score before and after scaling alongside the primary uncalibrated results, and the bounded label-blind leakage probe against full held-out scores with its 0.05 margin annotation – bounded probe only; no architecture-metadata claim. The benchmark contains 64 families, 4096 pair rows, and fixed training/development/held-out family partitions; model selection uses development families only. Primary intervals use source-family clusters; Brier/ECE have no clustered interval in the current receipt; Platt-scaled values are reported alongside and the uncalibrated scores remain primary; and the additive family-shuffle and label-permutation negative controls gate no claim. Positive deltas do not by themselves authorize a superiority claim, and no post-hoc model winner is selected. This is bounded endpoint evidence, not unrestricted semantic equivalence or application performance.

Finite quotient construction and source-family holdout

Source-level roles and endpoint labels are generated before inference; the visual separates an operational quotient under a finite intervention catalog from unrestricted semantic equivalence.



Finite quotient witness table

pair	declared role	observed endpoint	label source
pair-1	equivalent	4/4 equal	declared relation
pair-2	equivalent	4/4 equal	declared relation
pair-3	equivalent	4/4 equal	declared relation
pair-4	near miss	0/4 equal	declared relation

Interpretation boundary
Equality means agreement under the declared interventions, observation projection, and output relation. It is not unrestricted program equivalence or universal identifiability.

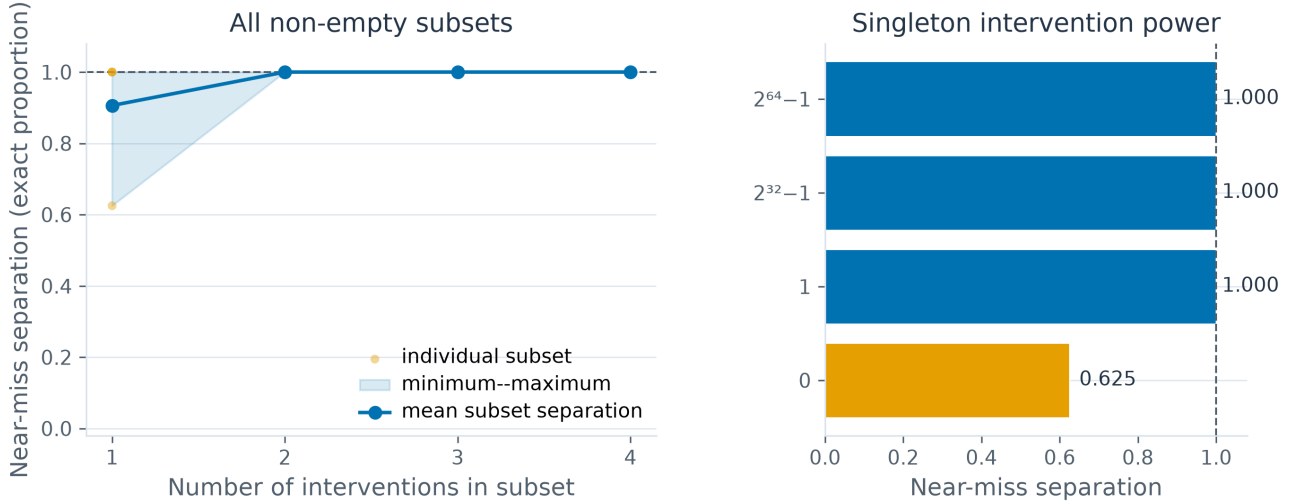
Figure 34: The finite quotient construction starts with 4 source-bound pair rows, applies 4 declared initial-state intervention values, compares normalized traces and the recorded output relation, and assigns the resulting benchmark label before fitting. The quotient classifier receipt contains 64 source families and 16 held-out families, while equivalent transformations, one-change near misses, and non-separating controls remain distinct roles. The table is a provenance view of label construction, not a learned semantic taxonomy. Equality means agreement under the declared intervention, observation, and endpoint contract; it does not mean unrestricted program equivalence or universal identifiability.

$$\begin{aligned}
\text{Sep}(S) &= \frac{1}{|\mathcal{N}|} \sum_{p \in \mathcal{N}} \mathbf{1}[\exists u \in S : e_{p,u} = 0], \\
\text{Cons}(S) &= \frac{1}{|\mathcal{E}|} \sum_{p \in \mathcal{E}} \mathbf{1}[\forall u \in S : e_{p,u} = 1].
\end{aligned} \tag{37}$$

where $\mathcal{E}$ and $\mathcal{N}$ are the declared equivalent and near-miss sets. A subset is *fully separating* only when both quantities equal one. The producer enumerates all 15 non-empty subsets over 16 pair outcomes, retains the tied minimum separating sets, and keeps the one-state negative control visible. The subset enumeration is visualized in fig. 35. The receipt is `../output/data/intervention_power.json`; its minimum fully separating cardinality is 1 with 3 tied set(s). Under the deliberately insufficient one-state control, separation is 0.625 and 3 near-miss families remain unresolved.

This construction is exact for the finite matrix and cannot by itself identify the underlying semantic quotient. In particular, a singleton separator may be an artifact of the chosen interventions, while an unresolved control can be a useful counterexample to an underpowered catalog. No intervention subset is allowed to be chosen after inspecting a model prediction, and the promotion status remains fail closed.

Finite intervention power in the declared quotient catalog



Exact enumeration of 15 subsets over 16 pair outcomes; no population uncertainty or universal identifiability claim is shown.

Figure 35: Exact finite intervention-power analysis for the source-family-disjoint quotient catalog. The left panel shows every one of the 15 non-empty intervention subsets as a point and the exact minimum–maximum envelope by subset cardinality; the right panel resolves singleton interventions. Across 16 equivalent and one-change near-miss pair outcomes, the smallest fully separating cardinality is 1 with 3 tied set(s). The all-zero negative control separates 0.625 of near-misses and leaves 3 unresolved. No sampling uncertainty is shown because this is exact enumeration of a finite receipt, not a population estimate or a semantic-identifiability theorem.

3.7 Measured sensing costs and finite policy certification

The active-sensing cost model has two layers. The primary layer is a deterministic vector

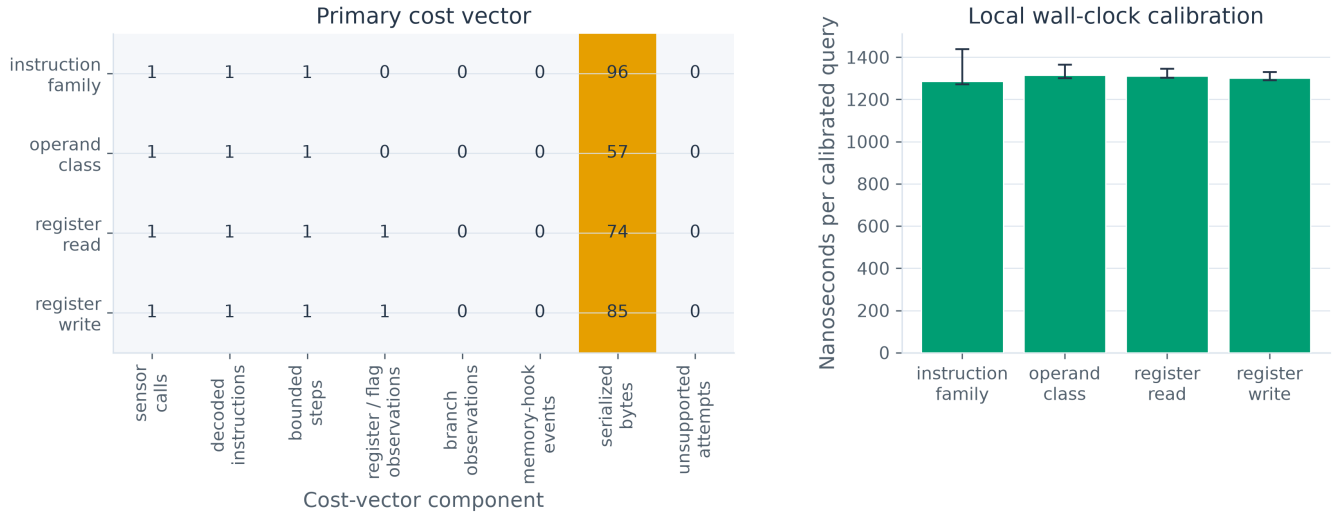
$$\begin{aligned}
c(a) &= (n_{\text{invoke}}, n_{\text{decode}}, n_{\text{step}}, n_{\text{reg/flag}}, \\
&\quad n_{\text{branch}}, n_{\text{memory}}, n_{\text{bytes}}, n_{\text{unsupported}}).
\end{aligned} \tag{38}$$

recorded for every query. It is machine-independent and auditable. The optional calibration layer executes a fixed matrix operation after warm-up and records median, IQR, 95th percentile, minimum, maximum, and raw repetition

values together with fixed-loop overhead, a deterministic-vector regression, and Python, operating-system, and package receipts. Wall-clock observations are not compared across machines without a matching calibration receipt. The source-bound receipt is `../output/data/sensor_cost_receipt.json`. The current sensor receipt contains 4 sensors and 16 exact finite certificates; its maximum reported optimality gap is 0.000.

For a finite state/action problem, the oracle value is obtained by exhaustive dynamic programming over every positive-probability sensor outcome. The expected-information component follows the classical view of an experiment as changing uncertainty under a stated prior, while the cost layer is separately audited [Lindley, 1956]. The implemented policy is certified only when its objective equals the oracle within the declared five-percent tolerance. The current policy is itself the exact finite dynamic-programming oracle, so its zero gap is a certificate of the enumerated benchmark—not of an infinite-horizon policy, a physical instrument, or a universal active-sensing law. The deterministic/calibrated comparison is shown in fig. 36. The same receipt evaluates fixed-first, greedy-information, seeded-random, and abstaining policies under the same finite outcome tree. Their regret and deterministic cost vectors are comparison diagnostics; the oracle is an inadmissible reference and the non-oracle policies receive no certificate unless they independently satisfy the declared tolerance.

Measured sensing costs and finite policy certificates



Eight deterministic components; exact cell counts and log-scaled shading. Timing bars: medians with min-max whiskers over 5 repetitions $\times$ 64 trials.

16 exact dynamic-program certificates; optimality is scoped to the frozen finite sensor space.

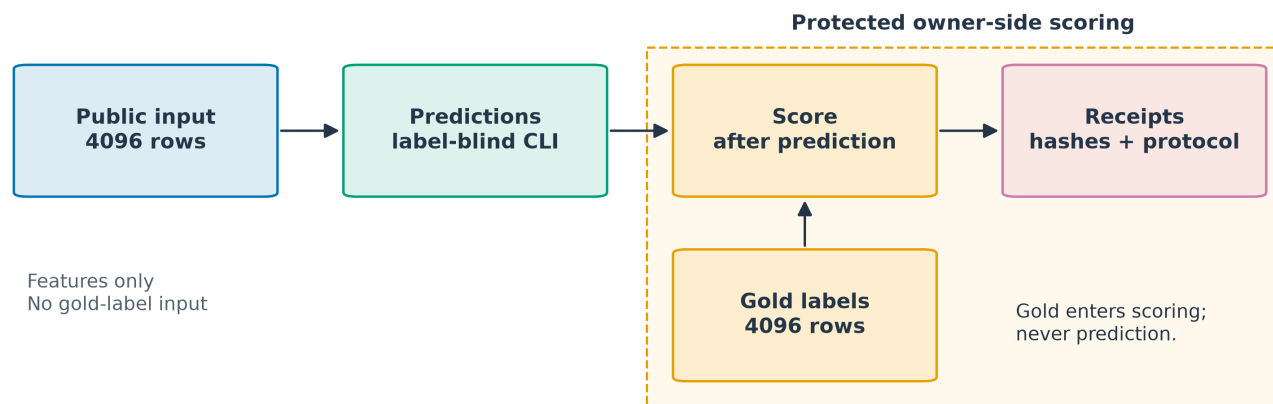
Figure 36: Two-layer sensing-cost receipt for the finite active-sensing benchmark. The left panel shows all eight deterministic cost-vector components as exact cell labels with shading used only for magnitude; the right panel shows machine-specific wall-clock medians and min-max whiskers in nanoseconds per query. The deterministic units are sensor calls, decoded instructions, bounded steps, register/flag observations, branch observations, memory-hook events, serialized bytes, and unsupported attempts. Timing uses the recorded calibration repetitions and trials, and the receipt also stores IQR, p95, overhead, and regression diagnostics. 16 exact dynamic-program certificates are scoped to the frozen finite state/action space; no cross-machine cost law is implied.

3.8 Frozen blind replication harness

The local replication interface is a CLI rather than a notebook state. Its metadata follows the stewardship aim of making research objects findable, accessible, interoperable, and reusable [Wilkinson et al., 2016], while the local receipt `../output/data/replication_manifest.json` makes the narrower claim of deterministic repository replay. The generator creates the deterministic public fixture/source input, validation checks all hashes and the protocol digest, each model emits predictions, and a separate scoring mode joins predictions to the gold labels. The public manifest

contains `record_id`, typed sequence, and feature values only, in deterministic digest order; the gold file contains labels and family/split metadata and is held outside the public input directory. The bundle currently contains 4096 public rows and 4096 gold rows, with label separation reported as `yes`. The public protocol, toolchain receipt, and offline instructions are part of the hashed public bundle (`c35b607c0d750405fc2ae901b1ba4b97b40d18e03cebbf7f3ac5cff5e2ea88ad`). Replay is local and no-network; the protected receipt separately records the current endpoint as `not_yet_observed` in the `not_observed` scope. The receipt retains separate model (`not_observed`), protected-score row-count (`not_observed`), and network-disabled (`not_observed`) fields. A completed receipt requires observed values for these fields and a separately controlled-run attestation. Replay of the frozen interface is evidence about the frozen interface, not an independent physical-machine run or an independent reimplementation. The separation contract is stronger than omitting a visible `label` key. Public record identifiers are derived only from public sequence/features, and the validator rejects any forbidden family, split, variant, or quotient field in the public rows. Exact public/gold row counts, protocol hash, input hashes, and prediction identifiers must match. A changed source or protocol hash therefore invalidates the receipt rather than silently producing a new benchmark under an old claim. The stable commands are `generate-public`, `verify-public`, `run-model`, `score --gold-path`, `verify-separation`, `replay --offline`, `write-replay-receipt`, `ingest-replay`, and `write-receipts`; the label-blind model path never opens the private gold directory. Receipt ingestion requires an explicit separately controlled-run attestation and a protected scoring boundary, so a local rerun cannot promote itself. The data flow is summarized in fig. 37.

Local replication harness and protected scoring



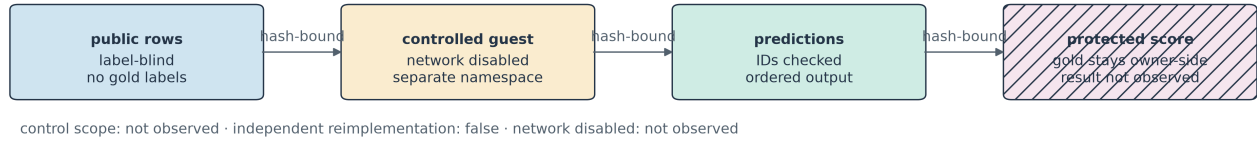
Stable IDs, ordering and input metadata carry no target label; hashes invalidate stale replay.
This schema does not establish that a separately operated or independent physical replay occurred.

Figure 37: Public/private provenance boundary for the frozen no-network replication harness. The diagram explicitly encloses the 4096-row gold-label surface inside a protected owner boundary, while label-blind public rows, predictions, and hash-bound receipts remain on the replay path. Identifier validation and protocol hashes are shown as gates rather than as scientific outcomes. Counts are deterministic and no sampling uncertainty is applicable to this manifest audit. Gold labels feed protected scoring after prediction and never the model input. This schema alone does not establish that a separately operated replay occurred; a retained receipt is required. A same-host controlled guest also does not establish independent physical replication or independent reimplementation.

The controlled replay audit (fig. 38) is included because a reproducibility package must make failure inspectable. When an independent receipt is ingested, the protected score is computed once over the frozen interface, and the audit retains whatever discrimination it shows as a bounded endpoint rather than converting it into a favorable claim by changing the model, labels, or endpoint. The guest boundary attests to the separately controlled-run procedure required for ingestion, while the protected scoring boundary confirms where labels were available. The recorded replay status is `not_yet_observed`; controlled replay evidence is not independent physical replication or independent reimplementation.

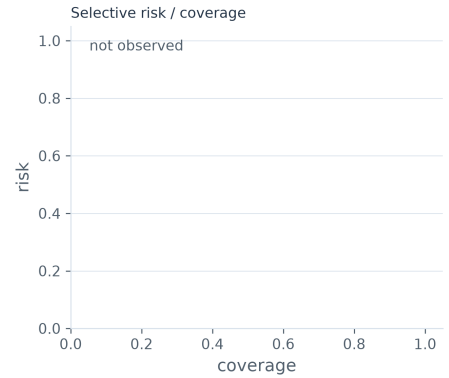
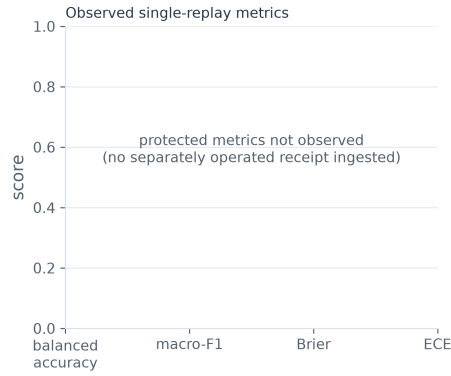
Controlled replay failure audit

Protected scoring is not observed. The diagram records the required boundary without inventing numerical results.



Protected confusion counts

Protected counts not observed
No separately operated receipt ingested



rows = not observed · false-equivalence rate = not observed · near-miss separation = not observed

Figure 38: The controlled replay audit is defined over the frozen public replay bundle: label-blind rows cross a no-network guest boundary, predictions return through identifier validation, and protected gold-label scoring remains on the owner side. No independently replayed receipt has been ingested; protected counts, balanced accuracy, macro-F1, Brier, and ECE are not observed. The confusion, reliability, and selective-risk panels therefore carry explicit not-observed statuses rather than measured values, and no substitute score, rerun, or expected value is imputed. When a receipt is ingested, this caption will report the protected result and retain it whether favorable or poor; a poor result is kept as a reproducibility and failure-boundary record rather than replaced by a favorable rerun. The audited boundary is a same-host controlled guest run, not an independent physical-machine replication or independent reimplementation.

4 Results

4.1 Prospective inference from executable GNN models

The sensor experiment tests acquisition order as well as numerical inference. Both fixtures enter the first decision with the same missing-observation state, prior, model, and policy distribution. The Agent chooses the dynamic sensor before its fixture-specific result is available. The next posterior separates the fixture hypotheses (fig. 39); the fixture names in the figure are evaluation labels and are not supplied to policy inference. A/B/C/D/E interventions, fixed-sensor and uniform-policy controls, and independent finite calculations are recorded in `output/gnn/controls.json`.

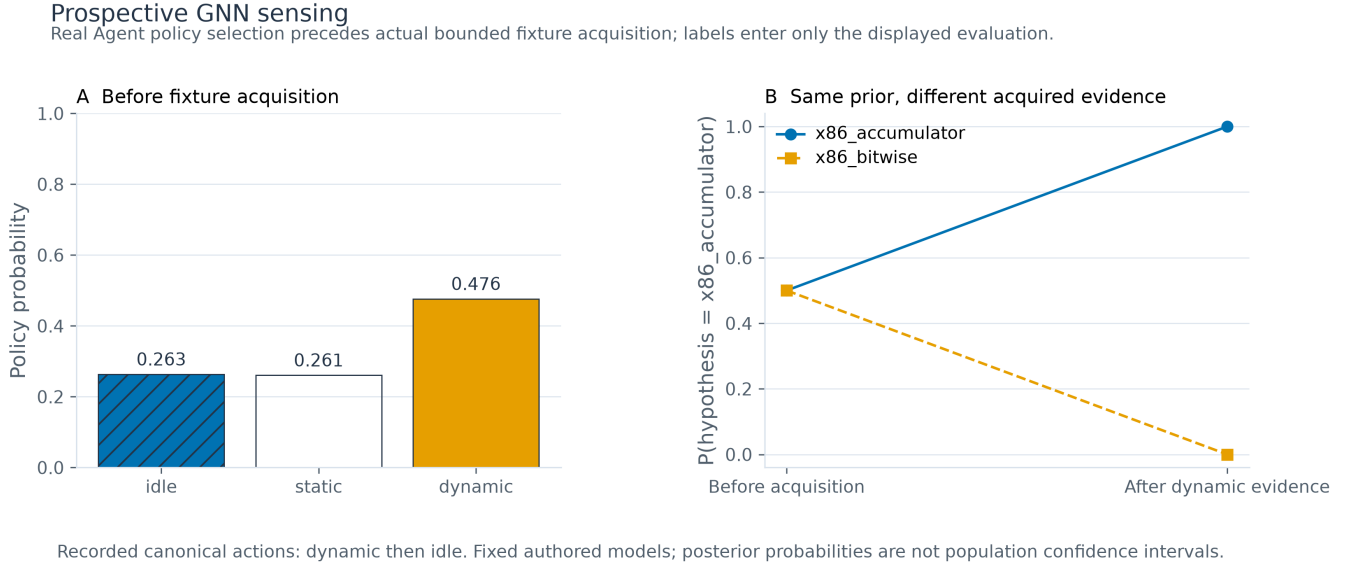


Figure 39: Prospective sensing on two transparent fixtures under the same fixed GNN model. Panel A shows the first policy probabilities: idle = 0.263, static = 0.261, dynamic = 0.476. The deterministic Agent selects dynamic before acquisition. Panel B shows the probability of the accumulator hypothesis before acquisition and after the first dynamic result, separately for the accumulator and bitwise fixtures. The two runs share the initial prior and first decision; only acquired evidence separates their subsequent beliefs. Lines connect two recorded belief states, not a continuous learning trajectory. These probabilities condition on authored likelihoods and illustrative dimensionless utilities; they are not uncertainty intervals, discovered semantics, or an external performance estimate [Friedman, 2026, Heins et al., 2022].

This result establishes that the declared model is parsed, exercised, and linked to prospective acquisition on the owned fixtures. The hand-authored likelihood deliberately makes dynamic evidence informative. Neither the resulting near-certain conditional belief nor agreement with an independent calculation measures generalization to unseen programs. The separate exact dynamic-programming study below evaluates a different finite decision problem; its optimality certificate is not a certificate for this predictive Agent.

4.2 Trace and comparison diagnostics

The canonical fixture `x86_accumulator` produces 4 decoded instructions and a bounded Unicorn trace with final `RAX` = 5. Its inference stream contains 4 categorical observations and 4 finite pymdp comparison diagnostics, one per observation, shown in fig. 42. The native RxInfer `gaussian event code model` records 8 finite Bethe Free Energy values with final value 8.237276 on Julia 1.12.7 and RxInfer 5.4.0, as shown in fig. 41. The final most likely pymdp latent class is `accumulator update`, reflecting the terminal observation history rather than an externally asserted semantic label; the posterior beliefs are shown in fig. 40.

The native package also executes the categorical phase-HMM companion `categorical phase hmm`, recording 8 finite iterations and final Bethe Free Energy 5.619808 in `../output/rxinfer_categorical/categorical_diagnostics.json`. This is a separate, fixed-parameter factor graph; its free-energy scale is not compared with the Gaussian spine.

Its iteration trace is flat because the structured categorical chain reaches its exact fixed point in the first sweep. The main manuscript’s fig. 13 displays actual native runs on successive observation prefixes. For this normalized tree model, the resulting Bethe free energy agrees with the independent scaled-forward value $-\log p(o_{1:t})$. Adjacent increments are conditional surprisal, $-\log p(o_t \mid o_{1:t-1})$, rather than an optimization trajectory. The unchanged repeated-iteration trace is retained in the native receipt. This distinction follows the factorization and message-passing schedule, and does not imply that arbitrary variational models converge in one iteration [RxInfer contributors, 2026a,b].

The independent bounded-semantics check supports the trace interpretation at the concrete level: 4 decoded instructions are covered, 0 executed instructions fall outside the declared subset, and the symbolic terminal RAX agrees with Unicorn (yes). The static control-flow projection contains 1 basic blocks and 0 edges. These are audited fixture facts, not a claim that a general x86 binary has been lifted or understood.

The stepwise comparison agrees for 4 of 4 executed steps, while the explicitly modeled flag bits agree for 12 of 12 comparisons. This tighter result is still a local instrument check: it prevents a later operation from hiding an intermediate transfer-function error, but it does not establish complete x86-64 semantics or semantic understanding.

The corresponding trace-refinement visualization is reported in the main manuscript’s fig. 7; it is kept as a single registry-owned figure so that the manuscript, HTML, and tagged-PDF outputs cannot assign competing labels to the same source-bound image.

The new formal and adapter receipts extend the evidence boundary without changing that interpretation. The Lean fragment is locally verified at status `locally_verified`, while the cross-architecture runtime fixture set reports 3 architectures, a common bounded result of 8, and an all-passed execution flag of `yes`. These are independent software and algebra checks on small declared surfaces. They do not turn the hand-authored phase labels into semantic states, and they do not establish x86/AArch64/RISC-V equivalence, compiler invariance, or external validity.

The memory instrumentation test is independently positive: the static `x86_memory` fixture contains 2 memory instructions, and bounded Unicorn emits 2 events comprising 1 write and 1 read at 1 distinct effective address(es). The observed store-to-load value and terminal result are retained in `../output/data/memory_evidence.json`; this is evidence that the instrumentation path functions on the transparent fixture, not a claim of general memory analysis.

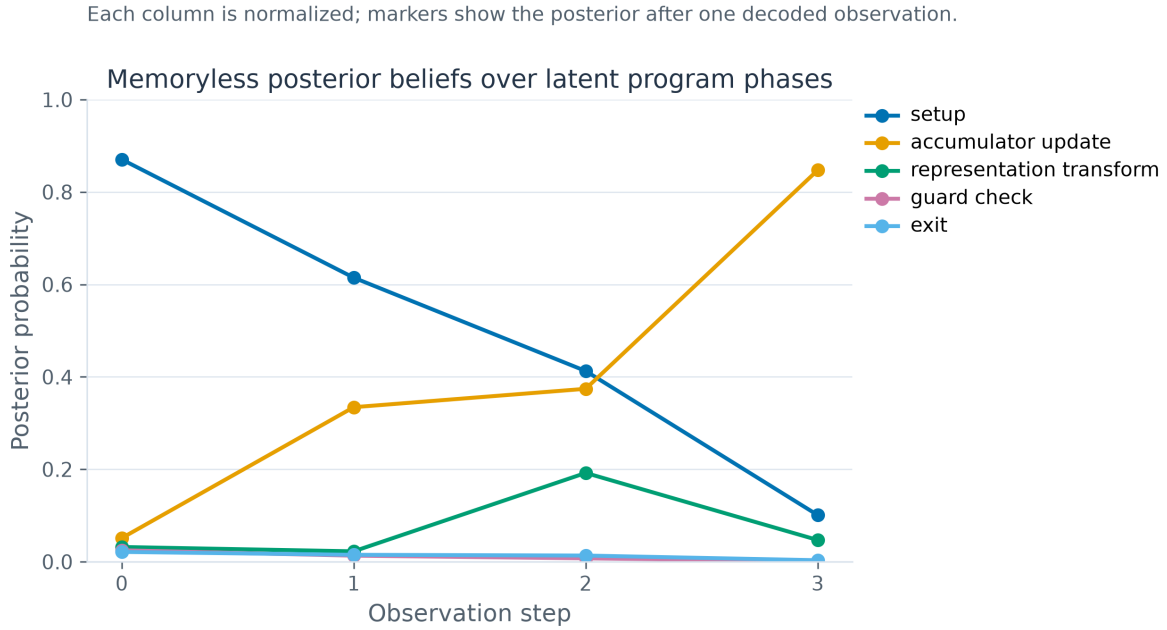


Figure 40: Memoryless posterior beliefs over the 5 hand-authored latent program phases at each of the 4 decoded observation steps. Each line is a normalized probability from the fixed comparison path; the visualization shows belief updates and does not supply independent phase ground truth.

The exact active-information analysis provides a second, mechanism-level result. The declared finite sensor family contains 4 typed-field channels and 16 horizon/cost rows; the receipt marks exhaustive branch enumeration as `yes` in `../output/data/active_information_frontier.json`. This is stronger than reporting one greedy policy score

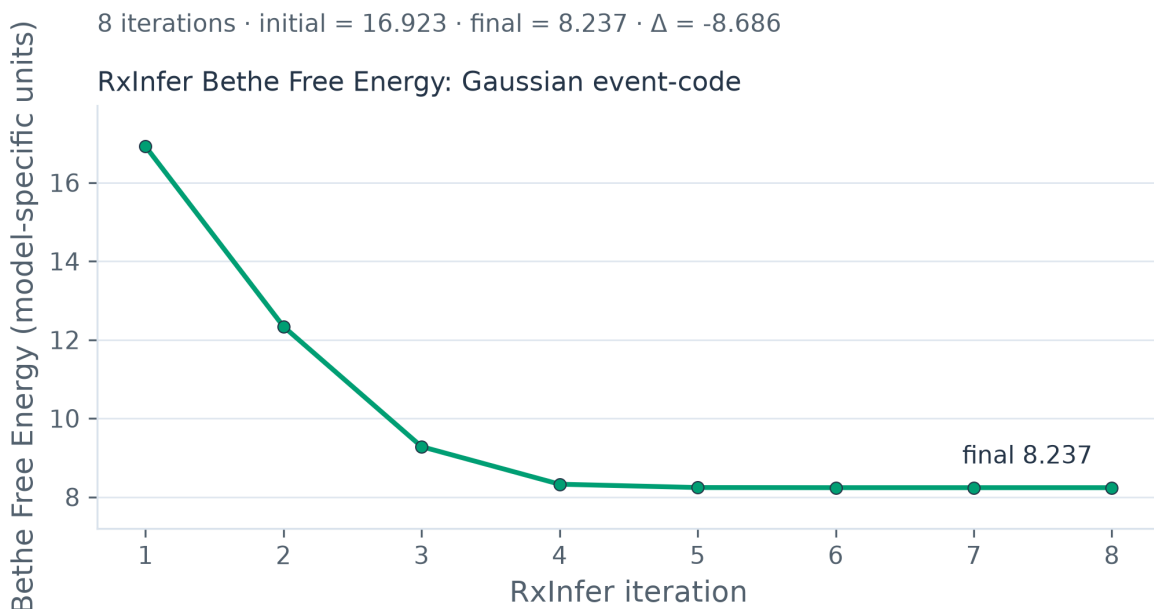


Figure 41: Bethe Free Energy values emitted by the native Julia/RxInfer backend for the Gaussian event-code model with GNN-specified Normal-Gamma priors. Points are backend iterations from the selected trace bundle; this is the authoritative RxInfer diagnostic for the implemented model, not the pymdp comparison objective and not evidence of semantic understanding.

because the bounded policy space is solved exactly, but it remains weaker than an information-per-cost law: sensor costs are proxy units, the channel is model-derived, and the horizon and sensor family are finite.

These results are useful as a working contract: every number in this manuscript is generated by the project code, and every figure—from the observation pipeline, temporal beliefs, model comparison, active-disassembly EFE, and inference diagnostics to the scholarship, claim, generative-adequacy, reproducibility, and validation panels—is written to `../figures/` before rendering. The generated caption ledger (`../figures/README.md`) and figure registry (`output/data/figure_registry.json`) document each figure’s inputs, statistics, citations, and accessibility metadata.

4.3 Measured phase decoding

We compare three decoders that share the *same* likelihood matrix and prior over 5 phase classes (chance accuracy 0.200): the **identity-transition** filter (an implicit identity transition that carries the previous posterior forward), a **memoryless** per-step decoder ($\text{argmax}_s D[s] \cdot A[\text{obs}, s]$, the strongest honest non-temporal baseline), and the current temporal HMM **forward filter** (the pre-declared headline decoder; Viterbi and smoothing are reported as secondary). Across the 10-fixture labelled corpus the mean decoding accuracies are 0.414 (identity-transition), 0.664 (memoryless), and 0.572 (temporal), over 56 labelled observation steps.

Two honest conclusions follow, neither of which is the naive “temporal wins.” First, the temporal HMM substantially beats the *identity-transition* filter — adding a left-to-right transition prior fixes the pathology where a peaked prior keeps the identity-transition filter stuck on **setup**. Second, and importantly, the *memoryless* decoder beats the temporal HMM on this corpus: temporal structure earns no net advantage here (temporal minus memoryless = -0.092), because the programs are short enough that each observation alone is strongly informative and the terminal **exit** phase is observation-underdetermined (it shares categories with earlier phases). Across the pooled corpus the temporal decoder recovers 0.607 of phases (descriptive Wilson 95% score interval $[0.476, 0.724]$) versus 0.696 (CI $[0.567, 0.801]$) for the memoryless baseline and the pooled identity-transition baseline (CI $[0.324, 0.576]$). The right reading is conditional — temporal modelling helps relative to an identity-transition filter, but a memoryless decoder is sufficient until programs are long or ambiguous enough that carrying belief forward pays off. The headline comparison uses hand-authored parameters and therefore measures model-structure value; the separate leakage-safe fit below is the only result presented as a generalization diagnostic. The accuracy metric is discriminative rather than green-by-construction: a shuffled-label control scores 0.221 and a uniform-belief control 0.200, both near chance.

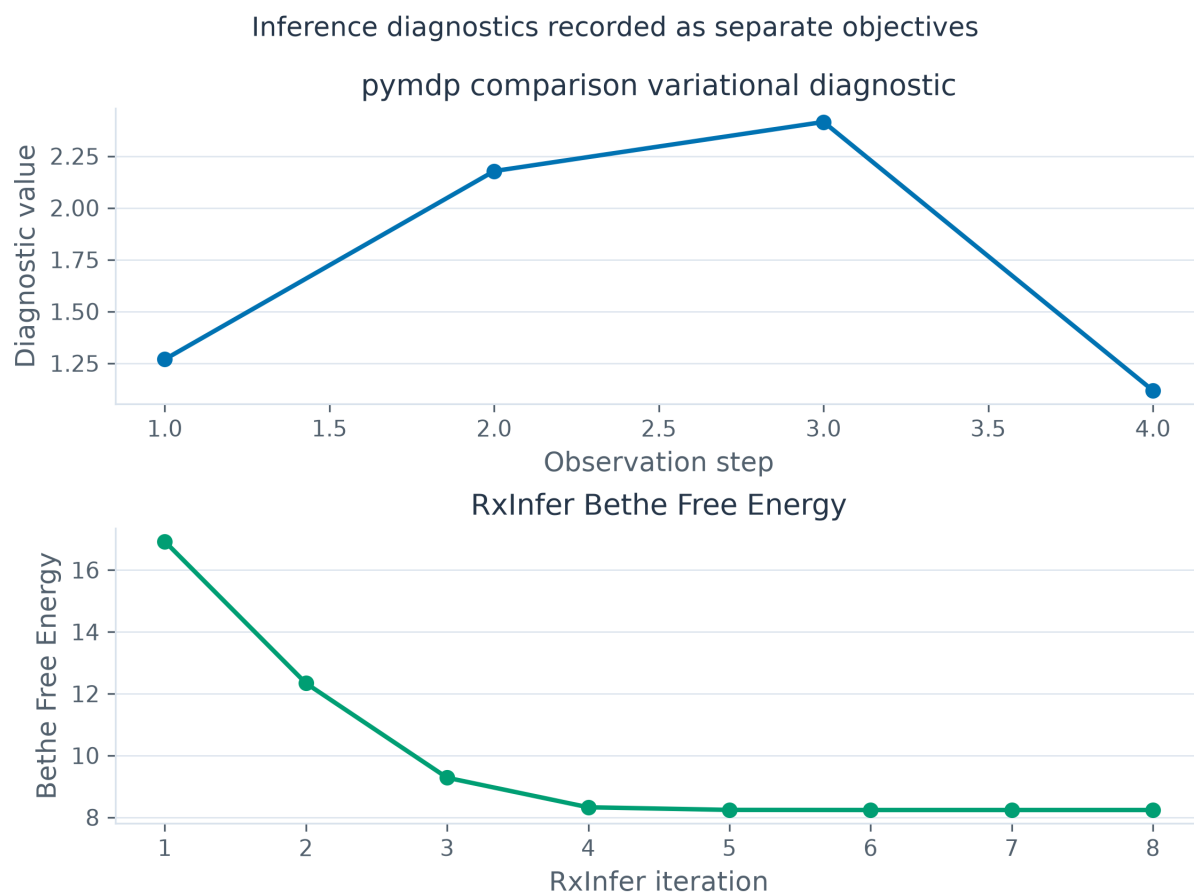


Figure 42: Two inference diagnostics kept on separate axes and scales: the pymdp comparison variational diagnostic above and the native RxInfer Bethe Free Energy below. The traces share the project input lineage but are distinct objectives and are not numerically interchangeable.

Per-fixture accuracy, calibration, free-energy convergence, and the full 4-decoder comparison (fig. 54) are written to the evaluation report and recomputed from disk by the validation gate. The Wilson intervals are descriptive per-step intervals, not independent fixture-level or population-level uncertainty estimates: steps within each fixture are dependent and the corpus contains only hand-authored toy programs.

4.3.1 Leakage-safe held-out estimation

The fixed-parameter comparison is complemented by a supervised estimation experiment with an explicit exclusion boundary. For each fixture, the smoothed categorical HMM estimates its emission matrix A, transition matrix B, and initial prior D from the other 10 - 1 fixtures only. The excluded fixture is then decoded without any parameter update. Across the 10 held-out runs, mean temporal accuracy is 0.667 (range 0.333–0.857) versus 0.761 for the fixed full-corpus reference. The lower held-out score is not a failure of the code; it is evidence that the tiny fixture set and phase-label distribution do not support a strong generalization claim. The exact training/held-out membership and fitted probability tensors are in `../output/data/held_out_model.json`. No claim of population performance or unsupervised phase discovery is made.

4.3.2 Observation-only ordered-HMM learning

The next research tranche removes the need to use phase labels while estimating the categorical parameters. The leave-one-complete-fixture-out experiment fits an ordered Baum–Welch model on the observation IDs from 10 complementary training sets, with a cap of 32 iterations per fit. The leakage contract is **yes**: labels appear only in the held-out scoring rows, not in the fit input. 0 of the fits reached the parameter-delta tolerance before the cap.

The observation-only EM fit reaches mean held-out accuracy 0.480, compared with 0.667 for supervised MAP fitting and 0.572 for the fixed reference. Its mean difference is -0.187 against supervised MAP and -0.093 against the fixed reference. The negative EM delta is the scientifically important outcome here: the observation-only estimator is real, reproducible, and leakage-safe, but it is not yet a competitive phase decoder on this short, highly constrained corpus. The ordered scaffold and reference initialization stabilize the latent-state coordinate system, while also limiting what can be called independent discovery. Training evidence histories and per-fixture predictions are retained in `../output/data/parameter_learning.json`.

The multi-start extension makes the remaining optimization ambiguity measurable. It retains 50 fits across the five declared starts for each holdout. Pooled across those fits, mean held-out accuracy is 0.363, the mean within-holdout accuracy range is 0.394, and mean pairwise predicted-sequence agreement is 0.323. These values are seed sensitivity summaries, not confidence intervals or a new sampling unit. Their spread is a limitation: the existence of a functioning observation-only EM implementation does not establish stable latent parameters, and no start is selected post hoc using the held-out labels.

The fixed-penalty Dirichlet-MAP comparator and posterior-predictive checks add a second robustness axis without changing the primary endpoint. The comparator is fit from the same label-free training sequences, while the predictive check reports observation-level structure rather than phase-label recovery. The receipt’s selection ledger records training-evidence decisions only; it does not turn a favorable predictive interval into evidence of semantic adequacy. If the ordered model remains initialization-sensitive, that instability is retained as a negative result. These fit diagnostics are visualized in fig. 43.

4.3.3 Typed observation expansion

The first roadmap expansion is now executable rather than aspirational. The source-bound typed schema retains 8 fields across 4 evidence families, with 7 static fields and one explicit dynamic-status field. The records are derived from Capstone detail and bounded Unicorn addresses, while the hand-authored phase sequence is consumed only by the supervised training/evaluation layer. The typed record artifact is `../output/data/typed_observations.json`; its leakage guard reports **yes** across 10 fixtures. The comparison reports 4 field-family ablations, so the result is not being inferred from a single arbitrary feature drop.

The held-out comparison is deliberately more informative than a claim of improvement. The collapsed categorical reference reaches mean accuracy 0.667; dropping dynamic status but retaining the static typed fields reaches 0.520; the full factorized composite reaches 0.520; and the balanced hierarchical score reaches 0.361. The full typed score therefore differs from the collapsed reference by only -0.147 and from the static typed ablation by 0.000. The

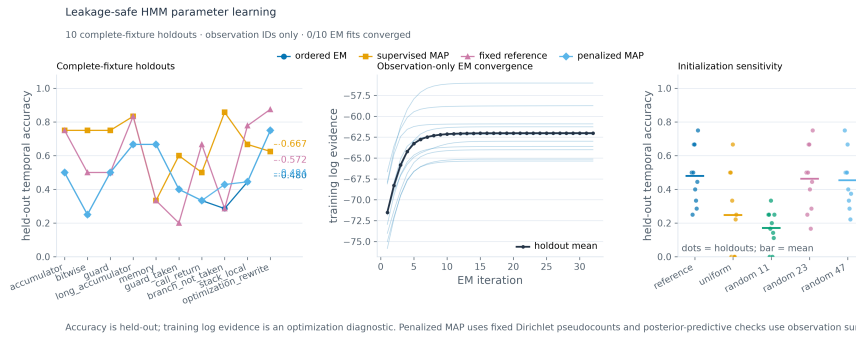


Figure 43: Leakage-safe ordered-HMM parameter learning. The left panel compares held-out temporal accuracy for observation-only ordered Baum–Welch EM, supervised MAP fitting, a fixed hand-authored reference, and a fixed-penalty Dirichlet MAP comparator on complete-fixture holdouts; the middle panel shows training log-evidence convergence for the primary EM fit and its mean; the right panel shows held-out accuracy for five deterministic initializations, with dots for holdouts and bars for within-start means. The receipt also records label-free posterior-predictive checks for the penalized fit. Phase labels are reserved for scoring and are not used in fitting; training evidence, predictive checks, and start-spread diagnostics are optimization/robustness diagnostics rather than held-out or population uncertainty [Dempster et al., 1977, Rabiner, 1989].

experiment is behaving as a useful measurement, including a negative result: the additional fields do not automatically create signal merely because the schema is larger. On the expanded corpus, the full field-wise score is -0.147 relative to the collapsed reference, and the balanced hierarchical score is not promoted as a replacement. This is not evidence that typed observations are unhelpful in general; it is evidence that this particular composite scoring rule is not yet adequate for the available fixture diversity. The initial register-dominated fixtures remain near-duplicate in some fields, while the expanded corpus adds one memory and one taken-path contrast. The field-group ablation and explicit masked-memory sensitivity are reported in the main manuscript’s fig. 5, with every training membership listed in `../output/data/factorized_observation_model.json`.

This result sharpens the scientific next step. A typed schema prevents information loss, but it cannot manufacture independent samples or justify conditional independence. The expanded corpus has a bounded memory-mode contrast and a static/dynamic path contrast, while still lacking the breadth needed for external validity. Masking the memory field changes mean accuracy to 0.562 (0.042 versus full factorization), which is a sensitivity diagnostic rather than a causal effect estimate. The result supports continuing to expand the fixture design with longer repeated phases, multiple memory addressing modes, and additional branch outcomes before attempting larger models; it does not support selecting the hierarchical score as a preferred model. The source-bound design inventory is reported in `../output/data/corpus_coverage.json` and the main manuscript’s fig. 6.

4.3.4 Statistical reporting contract

The reported quantities answer different questions and should not share an interpretation merely because each is numeric:

Quantity	Evaluation unit and interpretation	Limit
GNN policy and posterior probability	A declared policy or hidden-state hypothesis under the fixed authored model	Conditional model probability, not an empirical confidence interval
Temporal hit proportion and Wilson interval	Scored instruction steps nested within authored fixtures	Descriptive pooled accuracy; steps are dependent
Quotient balanced accuracy and macro-F1	Pair predictions on held-out transformation-family identifiers	Conditional finite-inventory endpoint; authored lineages cross splits
Family-bootstrap interval	Whole held-out identifiers resampled with their rows	Does not model dependence between shared lineages or model-refitting uncertainty

Quantity	Evaluation unit and interpretation	Limit
Paired randomization diagnostic	Complete model/baseline prediction vectors swapped within held-out identifiers	Requires exchangeability at that unit; retrospective analysis cannot establish superiority
Fixed-model adequacy interval	Same-length sequences simulated from fixed tensors	Model-checking distribution, not population or parameter-posterior uncertainty

Implementation agreement, model probability, descriptive performance, conditional randomization, and independent replication are separate evidence types. In particular, neither a large row count nor multiplicity adjustment creates an independent source sample [Cameron et al., 2008, Holm, 1979].

For decoder m , the primary accuracy is the pooled hit proportion

$$\hat{p}_m = \frac{H_m}{N}, \quad (39)$$

where H_m is the number of correctly decoded labelled steps and $N = 56$ is the common pooled step denominator. We use pooled accuracy for the headline comparison and report fixture-level means separately; neither should be read as an independent sample of binaries. The reported interval is the descriptive Wilson score interval

$$\frac{\hat{p}_m + z^2/(2N) \pm z\sqrt{\hat{p}_m(1 - \hat{p}_m)/N + z^2/(4N^2)}}{1 + z^2/N}, \quad (40)$$

with z set by the configured 95% confidence level [Brown et al., 2001]. Calibration is reported with equal-width reliability bins, expected calibration error, and Brier score; the bin counts remain visible in fig. 44. We do not attach p-values or claim external uncertainty because the corpus is hand-labelled, the headline comparison uses fixed parameters, and instruction steps within a fixture are correlated. The statistical purpose is auditability of a mechanism demonstration, not a population-level benchmark.

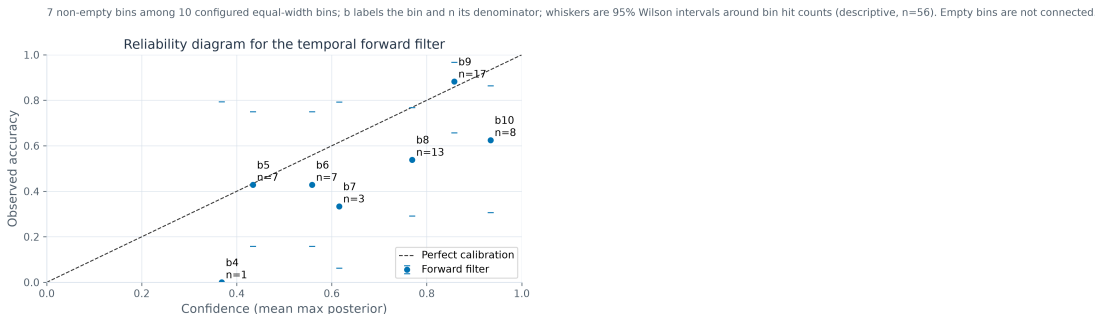


Figure 44: Reliability diagram for the temporal forward filter over 56 labelled observation steps. The plot shows 7 non-empty bins from configured equal-width confidence bins configured by the protocol; each point plots mean maximum posterior against observed accuracy; n labels the bin denominator, vertical bars are 95% Wilson intervals around bin hit counts, and the dashed diagonal is perfect calibration. Points below the diagonal indicate overconfidence for this fixture corpus [Guo et al., 2017, Brown et al., 2001].

Calibration is also compared across decoders. Reporting each decoder’s own per-step posterior (fig. 45) shows ECE tracking accuracy on this corpus: the **memoryless** per-step decoder is the best calibrated (ECE 0.106), the temporal model is middle (0.144), and the identity-transition filter is worst (0.337). This is stated honestly: the temporal model is neither the most accurate nor the best calibrated.

The decoding results above are classification metrics, summarized as a pooled confusion matrix in fig. 46. A complementary, principled *generative* comparison — the log marginal likelihood of the observed traces under each model (fig. 47) — shows **temporal** is the best *explainer* of the data: its pooled evidence (−99.882) is higher than the identity-transition (−108.142) and memoryless (−100.048) alternatives, a margin of 0.166 over its best alternative.

Decoder calibration metrics
Best by ECE: memoryless; worst by ECE: identity_transition. n=56 labelled steps; panels use independent y-scales because ECE and Brier are distinct metrics.



Figure 45: Decoder calibration on the labelled corpus. Separate panels report expected calibration error and Brier score for memoryless, temporal, and identity-transition decoders; both metrics are dimensionless and lower is better. The ECE ordering identifies memoryless as best for 56 labelled steps and is descriptive of these hand-authored traces, not a deployment-reliability guarantee [Guo et al., 2017, Brown et al., 2001].

Cells show count and row percentage; rows are true phases and columns are predicted phases.

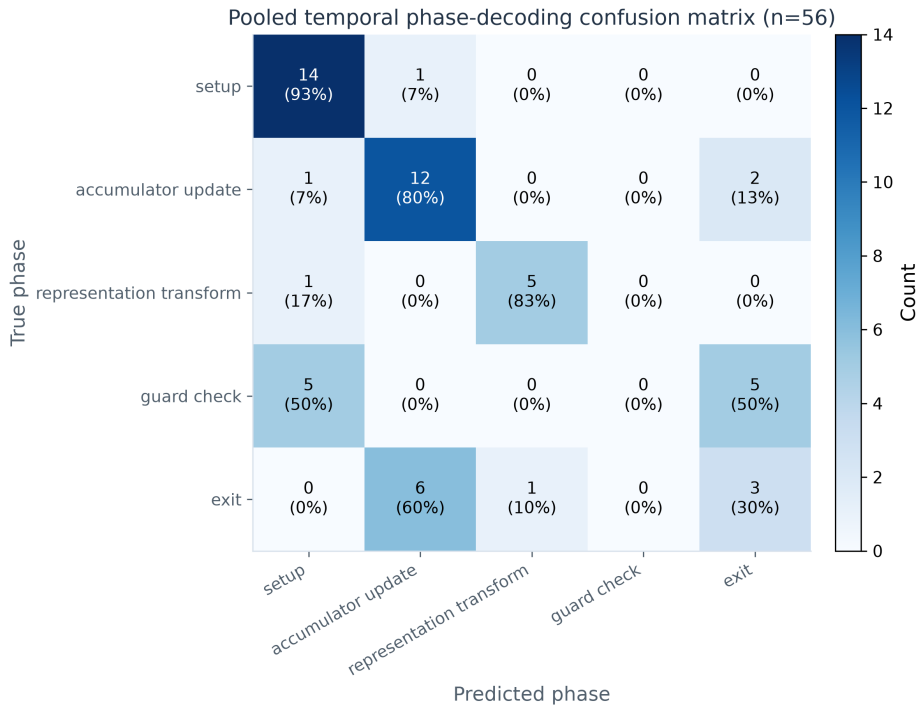


Figure 46: Pooled phase-decoding confusion matrix for the temporal forward filter across the labelled fixtures. Each cell shows the integer count followed by the percentage of its true-phase row; diagonal cells are correct recoveries and off-diagonal mass identifies confusable phase labels.

So the temporal model best explains the observed sequences even though the memoryless decoder is the better per-step classifier: the two comparison criteria (discriminative accuracy vs generative evidence) do not pick the same model, and we report both honestly.

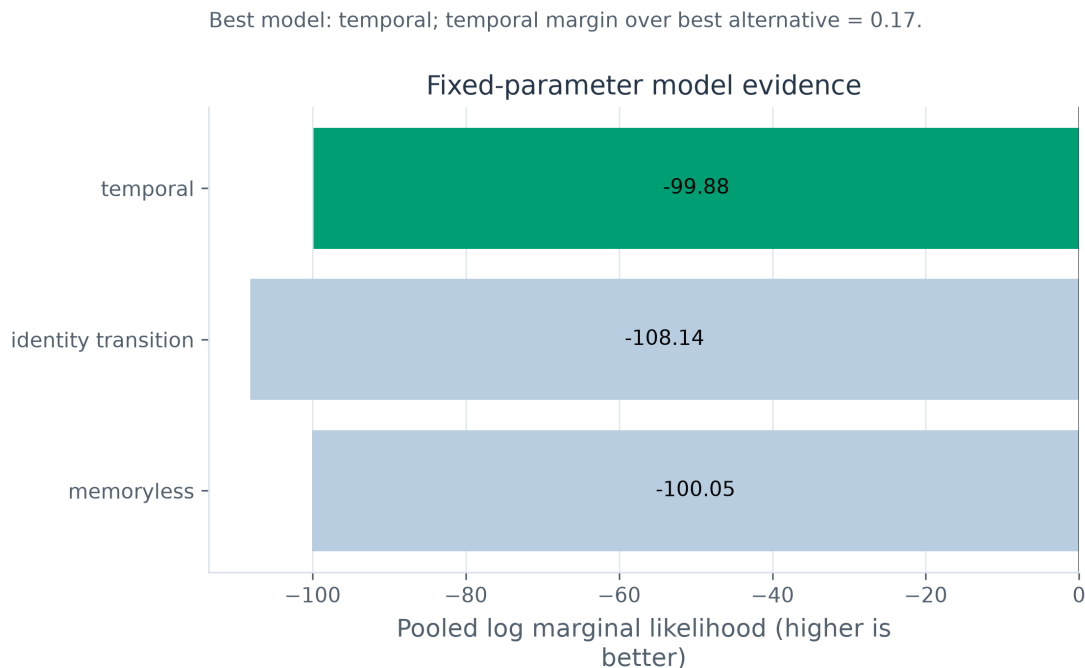


Figure 47: Pooled log marginal likelihood under three fixed-parameter generative assumptions: temporal HMM, identity-transition, and memoryless. Higher values indicate better in-sample predictive normalization for these labelled traces; they are not posterior probabilities of the models. The highlighted temporal model has the largest value, while the companion accuracy figures report the separate discriminative comparison.

4.4 Where temporal dynamics matter

The head-to-head result — temporal beats the identity-transition filter but not the memoryless decoder — has a precise, measured mechanism. The authored likelihood is nearly diagonal: each observation category peaks sharply on one phase, so a single observation already reveals the phase and transition structure adds little. To surface this, we blend the likelihood toward uniform (observation ambiguity), holding the corpus, transition matrix, and prior fixed. As ambiguity rises, per-step decoding collapses — at blend 1.00 the memoryless decoder falls to 0.266, while the temporal forward filter, using the left-to-right transition structure, retains 0.482. The temporal advantage over the memoryless decoder first emerges at blend 0.80. The result is honest in both directions: with the authored, informative likelihood there is no net temporal advantage, and a positive advantage appears exactly when observations become ambiguous. This is a mechanism demonstration on the same transparent fixtures (only the likelihood conditioning **A** is swept; **B** and **D** are fixed), not a claim about real-world observation noise.

The same mechanism appears in the *segmentation* dimension (middle panel of fig. 48): at the authored likelihood the memoryless segmenter already recovers phase boundaries well (boundary F1 0.854), but under full ambiguity its boundary recovery collapses to 0.000 while the smoothed temporal segmentation retains a boundary F1 of 0.714. Temporal structure rescues segmentation exactly when per-step decoding cannot.

The *active* layer behaves the same way (right panel of fig. 48): the realized-information-gain EFE heuristic reaches the confidence threshold in fewer steps than a fixed scan on 0.400 of fixtures, within an intermediate-ambiguity band (0.40 to 0.80), and the machine-checked never-worse result is **yes** (see `../output/data/active_selection.json`). The EFE ordering carries real value only when observations are *differentially* informative — neither when they are all equally informative (the authored likelihood) nor when none are (fully uniform).

The same artifact records a predictive-information policy for comparison. On the selected fixture it reaches the threshold in 1 steps under the predictive category contribution; this is reported as a policy diagnostic, not as a

corpus-scale gain. The one-step expected information of the authored observation channel is 0.667 nats, and the upper panel of fig. 55 keeps the realized and predictive EFE traces visibly separate.

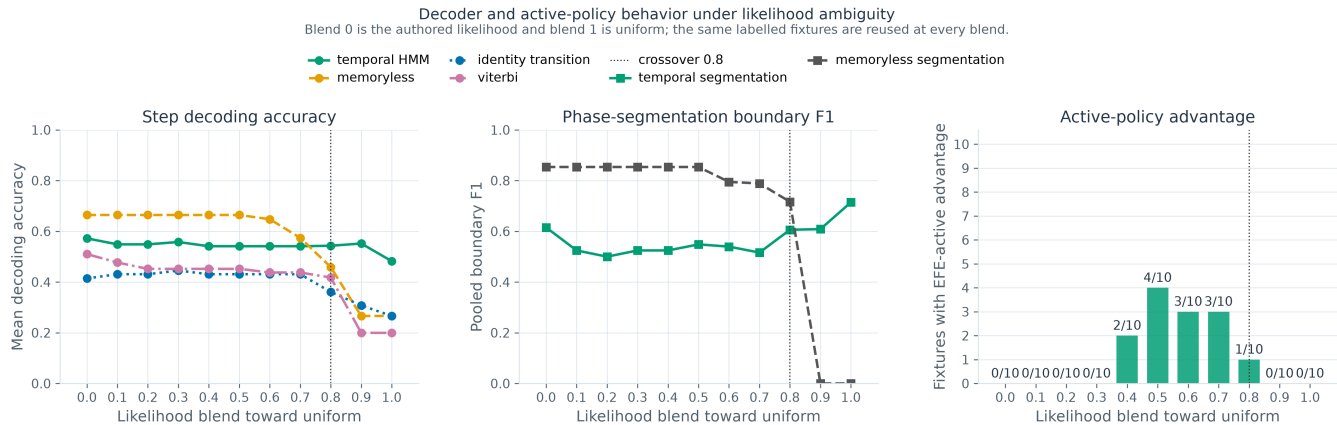


Figure 48: Three linked ablations on the same 10 transparent fixtures. Left: mean per-step decoder accuracy as the fixed categorical likelihood is blended toward uniform. Middle: pooled phase-boundary F1 for smoothed temporal and memoryless segmentations. Right: the number of fixtures in which the realized-EFE policy reaches the confidence threshold sooner than a fixed scan. Blend 0 is authored and blend 1 is uniform; the sweep demonstrates a mechanism under synthetic ambiguity, not a measurement of real-world binary noise.

4.5 How well we segment phases

Segmentation is not only self-consistent but measured. Decoding the smoothed forward-backward beliefs into per-step phases and comparing the detected phase boundaries against the hand-authored ground truth yields a pooled boundary F1 of 0.615 and a per-step accuracy of 0.589, available per fixture in `../output/data/phase_segments.json` and the evaluation report (`../output/data/evaluation_report.json`). The temporal filter and smoothing outputs are retained in `../output/data/temporal_inference.json`. fig. 49 makes this visual: for each fixture the top ribbon is the ground-truth phase sequence and the bottom ribbon is the sequence recovered from the smoothed beliefs, so a mis-segmentation shows up as a boundary mismatch between the two rows.

A second ablation isolates the contribution of the transition structure itself (fig. 50), matching the uniform-transition proposition: as the transition matrix `B` is blended toward a uniform matrix, the temporal decoder converges *exactly* to the per-step memoryless decoder — with `B @ posterior` uniform, the forward filter no longer carries state history — reaching equivalence by blend 0.8. The entire temporal-vs-memoryless difference is therefore attributable to the transition structure `B`. On this small corpus the peak mean accuracy occurs at a mild blend (0.680 at blend 0.3), an honest reminder that brute accuracy and the authored transition structure need not coincide.

4.6 Which phases are recoverable

Collapsing the pooled confusion matrix into per-phase metrics quantifies the recovery limitation that the manuscript cites qualitatively: the `setup` and `accumulator_update` phases are recovered at recall 0.933 and 0.800, the `representation_transform` phase has F1 0.833, while the `exit` phase (F1 0.300) and — hardest of all — the `guard_check` phase (F1 0.000) are observation-underdetermined. The macro F1 is 0.523 and the hardest phase to recover is `guard_check`; the per-phase profile is shown in fig. 52. The retained hard case is by design: in the optimization-rewrite fixture the `lea` rewrite shares the residual `other` observation category with unrelated instructions while its latent phase is accumulator update, so belief mass splits on that step and the designed ambiguity survives decoding. These are forward-filter decoding limits of the demonstrator, not claims about a complete binary-analysis system.

Likelihood information is uneven (fig. 51): `load_immediate` is most informative and `control_transfer` least, with an information range of 0.592 nats.

Pooled boundary F1 = 0.615; colors encode phase and each truth/prediction pair shares one fixture axis.

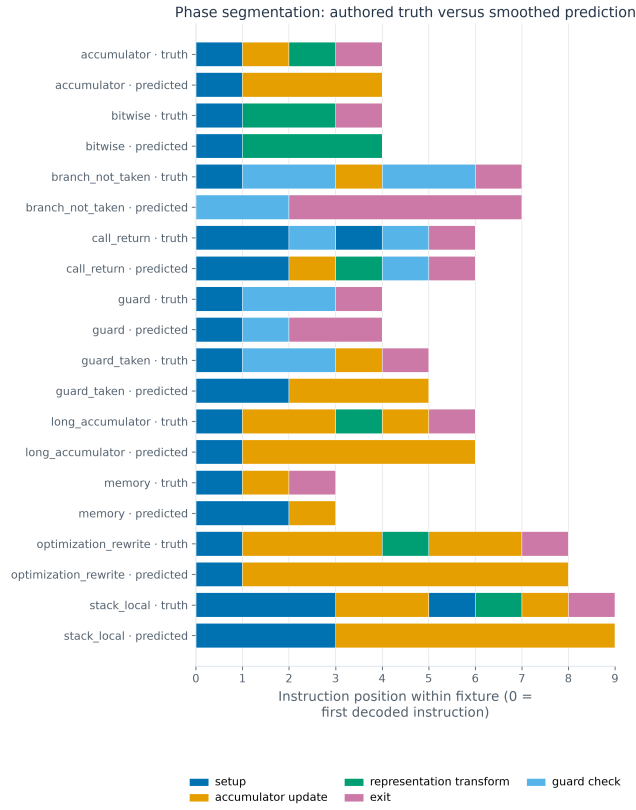


Figure 49: Per-fixture phase-segmentation ribbons. For each fixture, the upper row is the hand-authored phase sequence and the lower row is the sequence decoded from smoothed forward-backward beliefs. Contiguous runs share a color, so a boundary disagreement is a displaced color transition. The pooled boundary F1 is reported in the figure subtitle and computed against the authored labels.

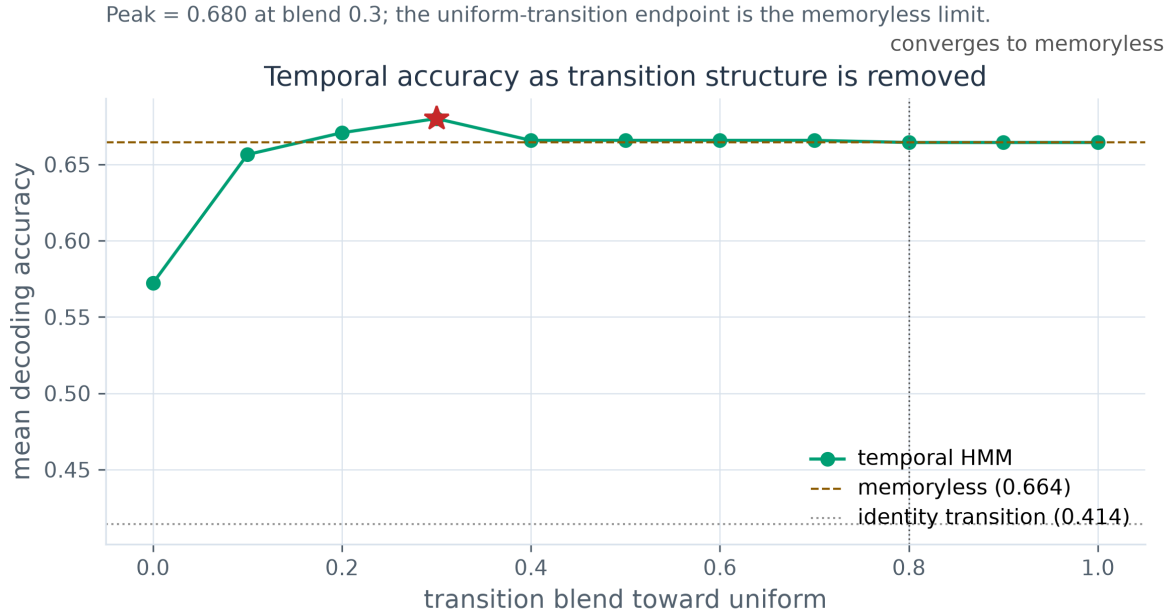


Figure 50: Transition-structure ablation across the likelihood blend grid. The temporal decoder’s mean per-step accuracy is plotted against constant memoryless and identity-transition references while B is moved toward a uniform transition matrix. At the uniform endpoint the temporal update collapses to the memoryless update, isolating the role of B; the marked peak records the accuracy optimum for this small fixed corpus rather than a claim that the peak transition is generatively correct.

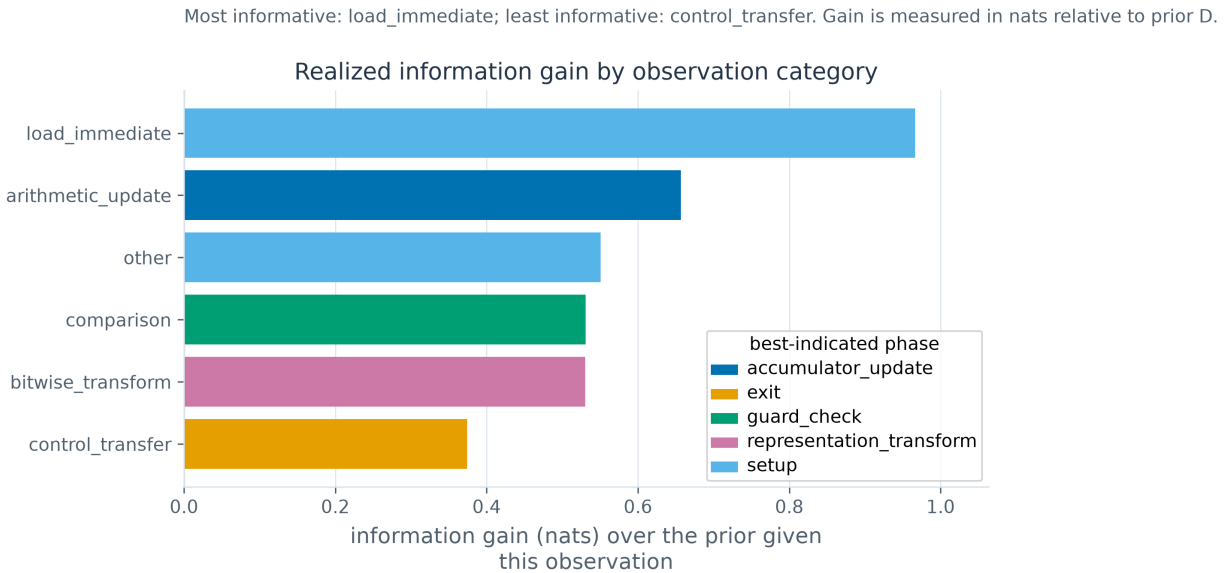


Figure 51: Realized information gain for each decoded-instruction category, sorted from largest to smallest and colored by the phase receiving the largest posterior contribution. The range is computed from the fixed likelihood and prior in evaluation_report.json; load_immediate is the largest-gain category and control_transfer the smallest. Because the plot uses the category actually revealed, it is a realized-information diagnostic rather than a full posterior-predictive expected-free-energy calculation.

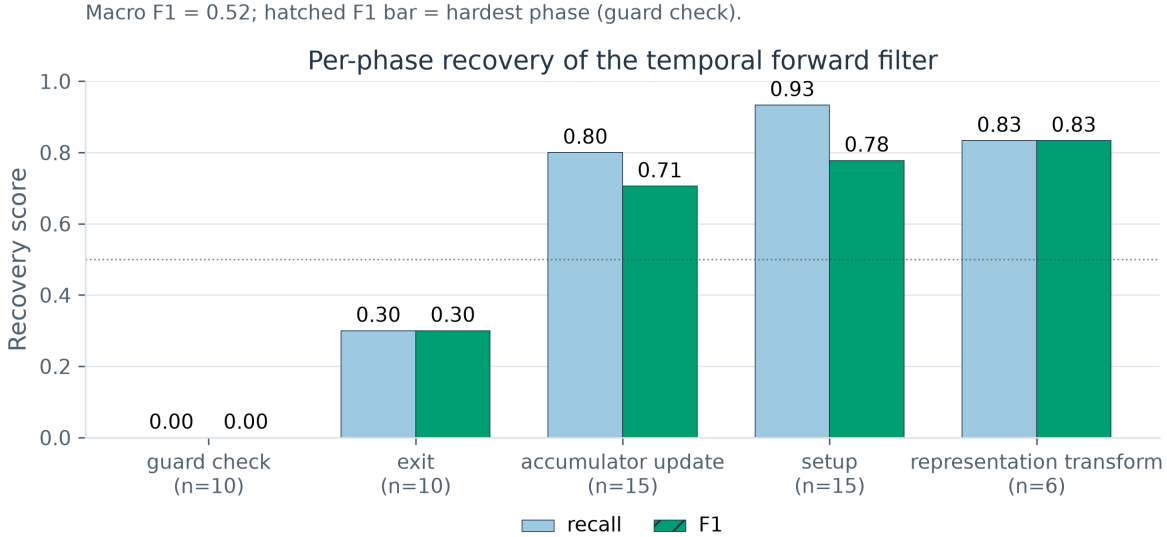


Figure 52: Per-phase recovery for the temporal forward filter, ordered by F1. Grouped bars report recall and F1 for each of the 5 hand-authored phase labels; the hardest phase and macro F1 are stated in the figure subtitle. These are corpus-specific label-recovery metrics, not evidence that the model has discovered universal program phases.

4.7 Robustness to the small corpus

Because the corpus has only 10 fixtures, pooled accuracies could in principle be driven by a single fixture. A leave-one-out analysis holds out each fixture in turn and recomputes the pooled accuracies over the remaining corpus: the pooled temporal accuracy ranges from 0.562 to 0.653 (span 0.091), stays above the pooled static (identity-transition) accuracy under every single-fixture removal, and remains well above the 0.200 chance floor. The leave-one-out sweep is visualized in fig. 53. This makes the small-corpus fragility explicit rather than hidden: the headline temporal result is not the artifact of any one fixture, and the span quantifies how much a single fixture moves the estimate.

The active-disassembly policy is reported honestly as a non-result at corpus scale. On programs this short the first observation already pushes the phase belief past the confidence threshold, so the expected-free-energy ordering ties the fixed left-to-right scan on all 10 fixtures while never underperforming it. The active *mechanism* — that an informative observation placed late in the natural order is selected earlier — is verified in the unit tests; the absence of a corpus-scale advantage is recorded as a measured fact rather than hidden. The complete evaluation surface is regenerated from `../output/data/evaluation_report.json` before template hydration.

4.8 Action-conditioned sensing

The action-conditioned experiment changes the evidence contract in a bounded way: the policy chooses a typed field query before the held-out field value is revealed. Across the 10 fixture holdouts, the active policy uses a mean of 1.000 actions under the common censored budget, compared with 1.000 for source-order fixed sensing. The active-minus-fixed contrast is 0.000 actions, so this tranche does not assert a universal active-policy budget advantage. The seeded-random baseline reaches the threshold on 0.600 of holdouts; the inadmissible oracle is shown only as an upper-bound reference. The action experiment is leakage-free by construction and by artifact check (**yes**): each fitted field model excludes the displayed fixture, and predictive ranking receives no realized field value. The complete policy records and revealed values are available in `../output/data/action_sensing.json`; the policy comparison is visualized in fig. 56.

4.9 Open-roadmap evidence tranche

The co-primary quotient endpoint is a portfolio evaluated on held-out family identifiers, not a post-hoc winner selection. The benchmark contains 64 families and 4096 pair rows under the training/development/ held-out split. The interpretable classifier reaches balanced accuracy 0.671 and macro-F1 0.654; the temporal probabilistic

10 held-out fixtures · observed temporal span = 0.09; bars are pooled over remaining steps.

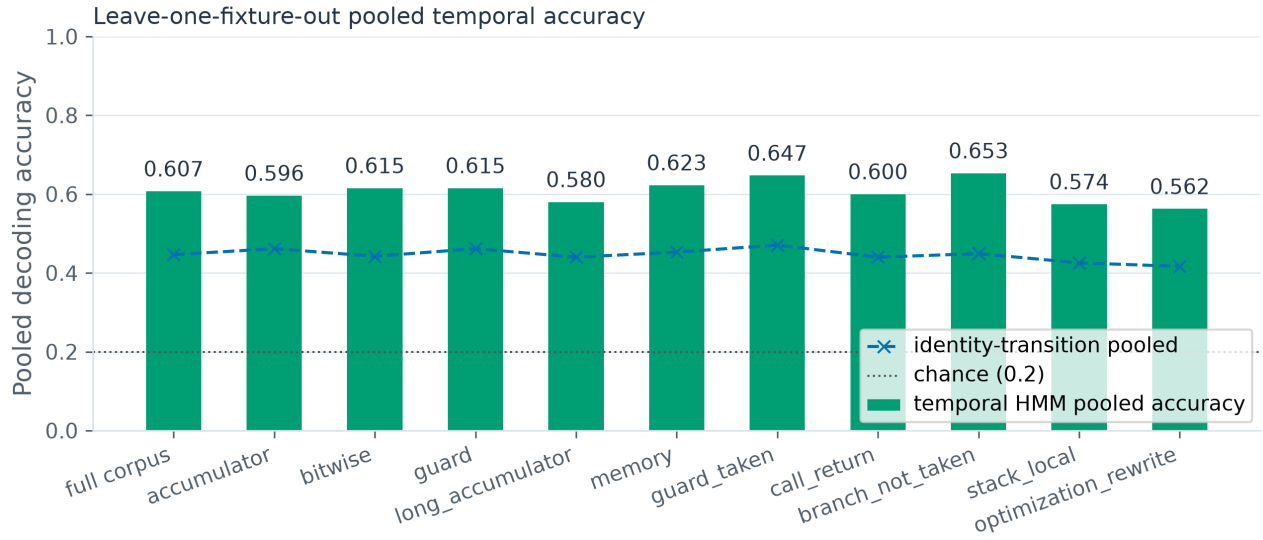


Figure 53: Leave-one-fixture-out stability analysis. Bars show pooled temporal accuracy on the full corpus and after removing each fixture; the dashed line is the pooled identity-transition reference and the dotted line is the configured chance floor. The span in the title is the observed range across the leave-one-out fits, making small-corpus sensitivity explicit.

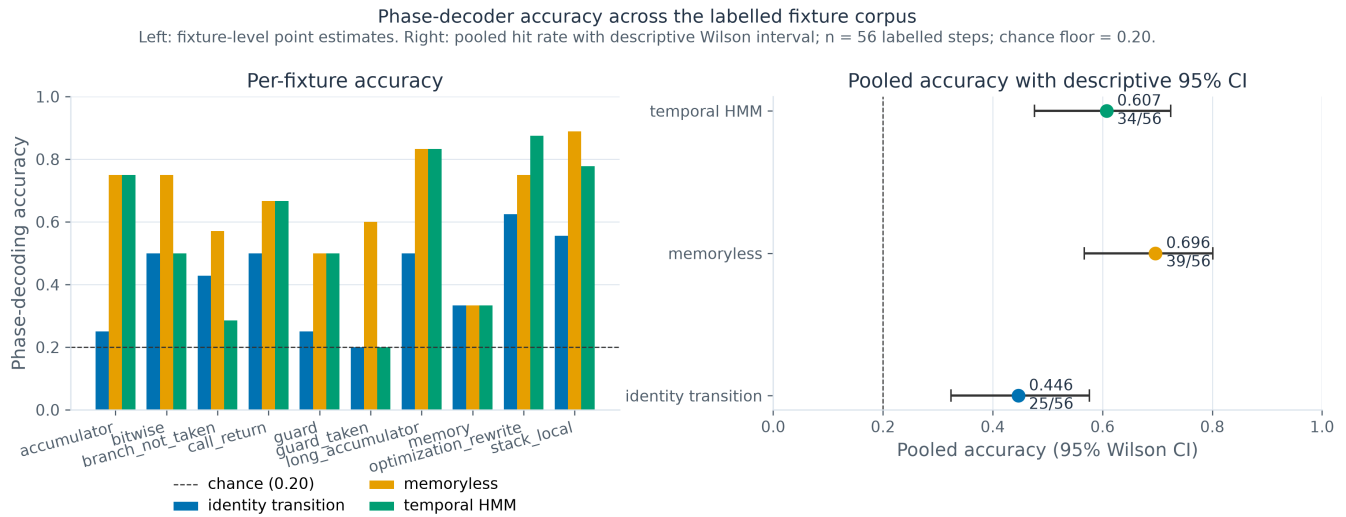


Figure 54: Phase-decoder accuracy across the 10-fixture labelled corpus. The left panel shows fixture-level accuracy for identity-transition, memoryless, and temporal decoders; the right panel shows pooled per-step accuracy with 95% descriptive Wilson intervals around hit counts over 56 labelled steps. Steps within a fixture are dependent, so the intervals are descriptive rather than population-level uncertainty estimates.

Active-disassembly trajectory
Realized EFE uses the decoded category; predictive EFE weights its information contribution by prior category probability.

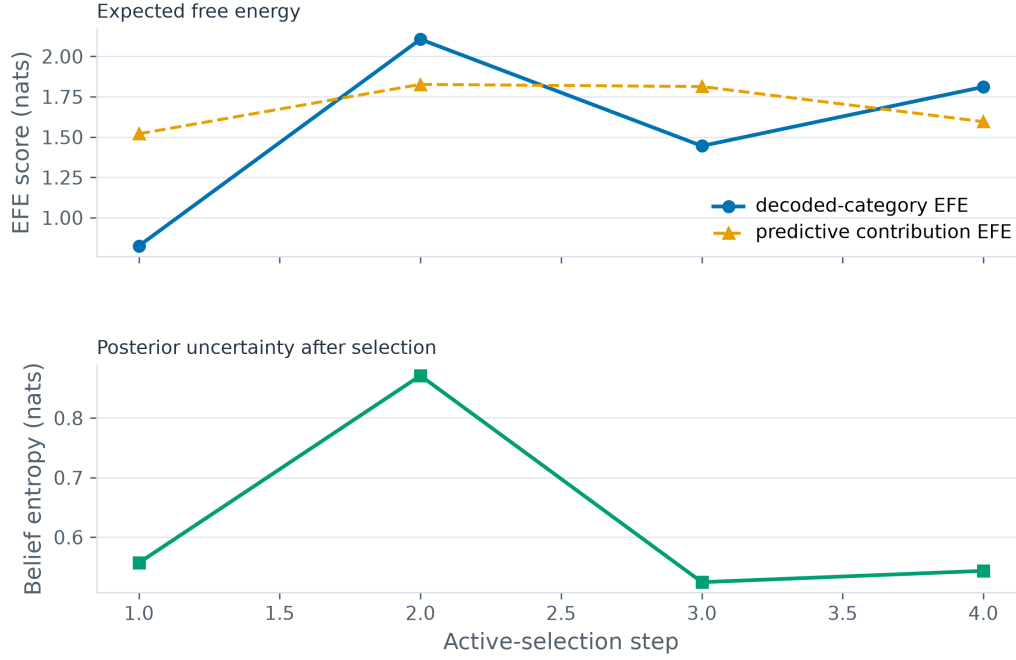
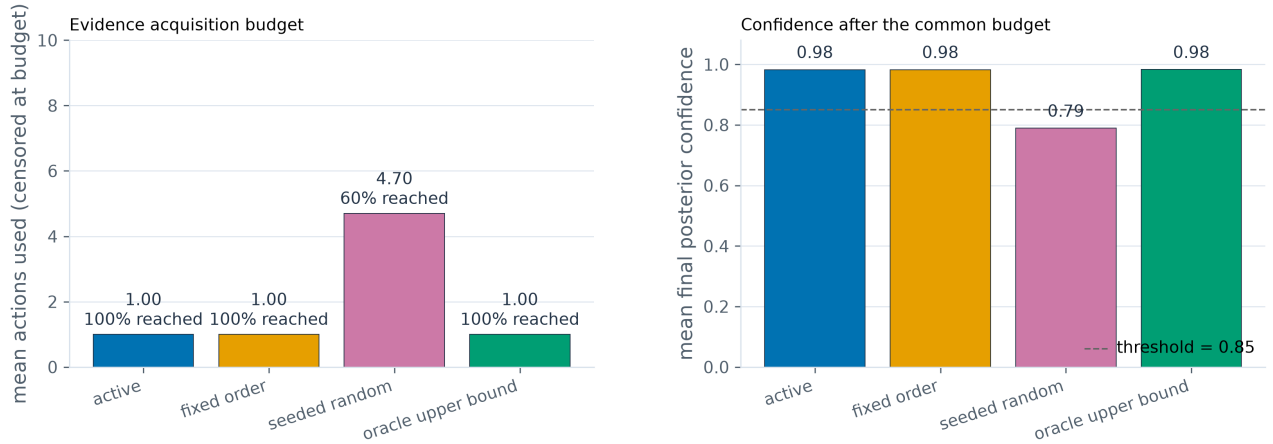


Figure 55: Greedy active-disassembly trajectory for the selected fixture. The upper panel reports both the decoded-category EFE score and its predictive information-contribution companion in nats; the lower panel reports posterior entropy on a separate axis. Predictive contributions weight a category’s realized entropy reduction by its prior predictive probability, and their sum is the one-step channel information gain. The machine-checked corpus-level never-worse result for the realized policy is true; a tie on short fixtures is not presented as an active-policy gain.

Action-conditioned typed-field sensing

10 held-out fixtures · 4 policies · equal normalized action cost · oracle is inadmissible



Active choices use predictive field likelihoods before the held-out value is revealed; random failures remain censored at the same budget.

Figure 56: Held-out action-conditioned sensing over typed assembly fields. The left panel reports the mean number of actions used before the configured confidence threshold, with failures retained at the common budget; the right panel reports mean final posterior confidence. Active choices use predictive training-only field likelihoods before the held-out value is revealed, fixed and seeded-random policies use the same candidate pool, and the oracle is an explicitly inadmissible upper bound. Equal normalized action cost is a mechanism contract, not a wall-clock measurement.

classifier reaches 0.775 and 0.772; the fixed one-layer tanh recurrent classifier reaches 0.485 and 0.340; and the typed-trace surface baseline reaches 0.674 and 0.657. Clustered intervals, Brier/ECE, confusion counts, and false-equivalence/false-distinction rates are retained in the source receipt. The receipt also fits 6 independent raw surface baselines and records 18 paired model-by-baseline comparison rows under one Holm family. With 16 held-out families and development scores that need not track test scores, these results do not establish superiority or learned semantics. They establish a reproducible, label-source-controlled endpoint with visible negative evidence, including a recurrent classifier that does not recover the endpoint reliably.

The portfolio’s robustness diagnostics are development-only and fail closed. Hyperparameters are selected on the development families only (`test_rows_used = 0`): the logistic comparator selects L2 penalty 0.001 and learning rate 0.2, while the recurrent comparator selects hidden size 32 and 8 epochs, with best development balanced accuracy 0.625. Ties use the frozen lexicographic first-maximum rule. Five deterministic starts are reported separately from the source-family bootstrap intervals. Platt calibration is fitted on development data; uncalibrated balanced accuracy and macro-F1 remain primary, and calibrated metrics gate no claim.

The within-family label-scramble diagnostic reaches balanced accuracy 0.597. It deliberately refits on scrambled held-out labels and evaluates on the same rows’ original labels. This is an in-sample perturbation diagnostic, not an additional out-of-sample prediction. Permutations preserve each family’s class totals, so they do not remove family-prevalence associations or certify absence of leakage. Comparisons exclude models at or below binary chance, as recorded in the receipt.

The bounded encoding/opcode-surface probe refits each model with reduced features and reports score differences of +0.004 (interpretable), +0.024 (temporal), and +0.031 (recurrent) against its full-feature endpoint. These are bounded refit diagnostics, with no architecture-metadata conclusion. The cross-split similarity ledger records 1280 family pairs and 128 declared near-duplicate siblings; the cross-encoding contract covers 61 of 64 families. These diagnostics qualify the endpoint and license no superiority, metadata-leakage, or semantic-discovery claim.

The real transfer and correspondence results are similarly bounded. The Python/Capstone/Unicorn receipt reports status `validated_bounded` for 3 transparent fixtures and 88 compared reference steps. The compiler receipt records a 184-cell source-by-optimization matrix over the compilers `clang`, `nasm`, `rustc` and the object formats `elf`, `macho`: 128 cells are validated static objects (`clang`, `nasm`, and `rustc`), while 56 cells are explicitly unavailable — 36 with `compiler_target_unavailable_or_compile_failed` (including every `clang` RISC-V target probe and all `go/zig` cells) and 20 with `fixture_source_not_embedded` — and validated architecture labels are limited to `aarch64`, `aarch64_linux`, `x86_64`, `x86_64_linux`. Because emitted objects were not executed and the separate licensed-source receipt is `underpowered`, no cross-compiler, application, or compiler-correctness claim follows. The figures make those boundaries visible rather than collapsing unavailable capability into a zero.

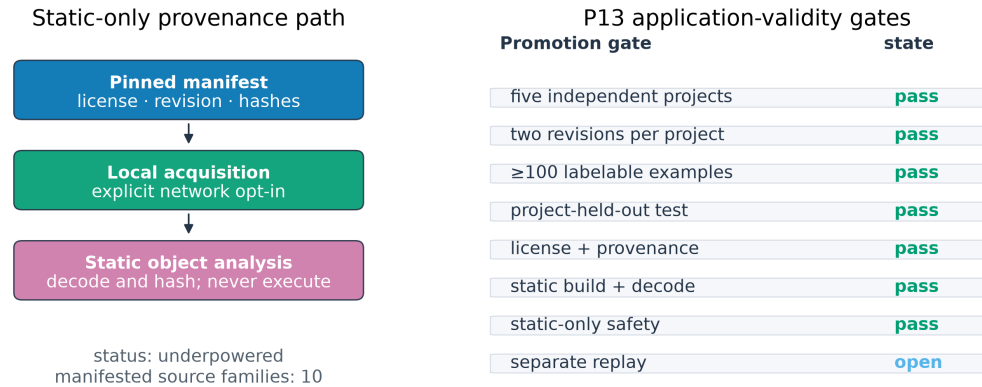
The generated Lean bridge is the formal counterpart: it covers 6 transparent fixtures with 20 generated machine-state theorems, all kernel-checked (`true`), and the axiom report is clean (`true`); this is a checked bounded-state foundation, not a correspondence theorem for the full pipeline.

The intervention-power analysis is a separate identifiability diagnostic rather than another model score. Exact enumeration of 15 non-empty subsets over 16 pair outcomes finds a minimum fully separating cardinality of 1, with 3 tied minimum sets. The one-state negative control separates 0.625 of near-misses and leaves 3 families unresolved. This is an exact result for the declared finite outcome matrix, not evidence that the intervention catalog is complete or that semantic identifiability holds outside the benchmark. The source-bound receipt is `../output/data/intervention_power.json` and the subset curve is shown in the main manuscript’s fig. 35.

The external-source receipt is `underpowered`: it records 5 manifested projects, 100 labelable examples, and a static-only receipt with 10 validated rows of 10; the provenance chain is visualized in fig. 57. The separate source-bound label receipt is `validated` with 100 accepted rows (40 lexical-equivalence witnesses, 40 contract-directed near misses, and 20 identity controls) and 0 explicit exclusions; the held-out project is `libyam1`. The external acquisition/build replay gate remains unpassed, so these source-contract and static results are not failed accuracy results and do not support application-facing effectiveness.

The accepted-row census makes the denominator and split topology explicit in fig. 58. It shows the role composition by project and the project/revision allocation used by the source-bound contract, including the held-out project marker. This is useful precisely because it does not pretend that a static object hash is a behavioral label: the figure reports what has a documented source-level contract, while excluded candidates remain visible in the caption and the receipt. The accepted rows therefore support a provenance and coverage statement, not a claim of application effectiveness or universal program equivalence.

Licensed-source provenance and application-validity gate



P13 pinned rows: 10; validated static rows: 10. No arbitrary external binary is executed. Open or unavailable gates remain status values; P15 controlled replay is a separate evidence surface, never a zero-valued performance result.

Figure 57: Licensed-source provenance and P13 application-validity gate for the separate external validation track. The left panel shows pinned acquisition, explicit network opt-in, and static-only compilation/disassembly; the right panel lists the immutable revision, license/hash, safety, project/revision/example, and separate-acquisition-replay gates. The current receipt is underpowered: 5 project(s) are pinned and 10 of 10 static rows decode successfully, with 100 source-contract rows accepted. Lexical witnesses, directed near misses, and identity controls are provenance labels, not universal behavioral equivalence. P13 remains distinct from the observed P15 controlled replay; no unavailable evidence is rendered as zero and no application claim is promoted by this diagnostic alone.

The observation-semantics plate makes the preceding boundary operational. It shows that the model input is a typed, label-blind record in which a missing field, an unseen categorical value, an unsupported operation, an unavailable capability, and a measured zero have different meanings. This supports the interpretation of the transfer and external-corpus panels as capability and provenance diagnostics: an unavailable target is retained as unavailable, not counted as a failed prediction. It also clarifies what the protected replay does and does not test. The public rows and returned predictions can be validated without exposing the gold labels, but the boundary alone does not establish absence of every side channel or make a same-host controlled run an independent replication. The visual is therefore a receipt-backed schema result, not another accuracy endpoint.

The measured sensing receipt reports 4 deterministic sensor vectors and 16 exact finite certificates. The maximum optimality gap is 0.000, but this is a certificate for the enumerated finite state/action space because the evaluated policy is the dynamic-programming oracle itself. Local wall-clock calibration is included for auditability and is explicitly machine-specific. It does not replace the deterministic cost vector or imply a cross-machine speed law.

The replication bundle has 4096 public rows and 4096 separated gold rows, with label separation **yes**. The current public harness is frozen and no-network. No independent replay has been observed: the recorded replay status is **not_yet_observed** in the **not_observed** control scope. The replay interface and the protected-scoring endpoint are frozen and ready; a prior same-host receipt records the status **submitted_pending_verification**, and no protected score exists to interpret. When an independent receipt is ingested, the protected score is computed over the frozen interface and reported with **not_observed** rows (**not_observed** positive/equivalence rows and **not_observed** negative/near-miss rows), balanced accuracy **not_observed**, macro-F1 **not_observed**, Brier score **not_observed**, expected calibration error **not_observed**, false-equivalence rate **not_observed**, and near-miss separation **not_observed**; such a score would be the result of one protected replay of the configured deterministic model, not a new scientific sample. A changed protocol/source hash invalidates the receipt, preserving the distinction between rerunning the same code, controlled replay, and independently implementing the method.

4.10 Theorem-or-countertheorem outcome

The strongest candidate tested in this tranche was intentionally vulnerable: that a single zero-state intervention would separate every declared near-miss family. The automatically generated counterexample is **zeroing_idiom**:

Source-bound external label coverage

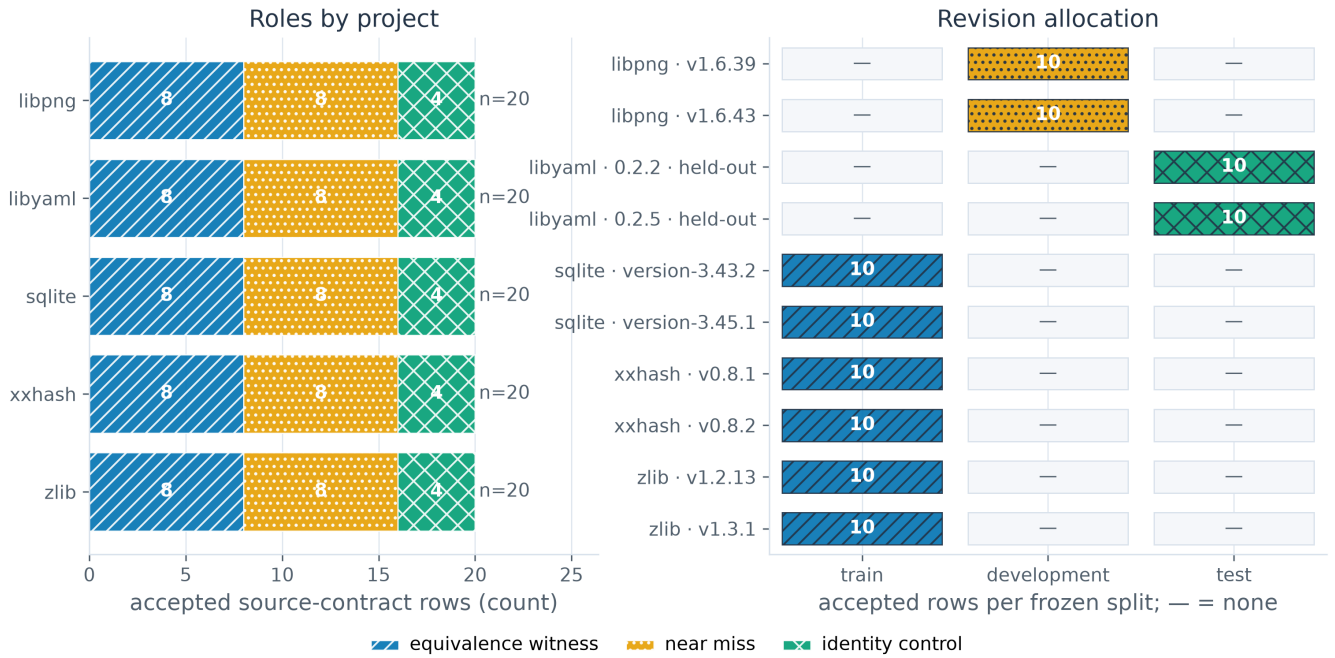


Figure 58: Source-bound external label-contract census for 100 accepted rows across 5 projects and 10 project revisions. The left panel counts source-defined equivalence witnesses, directed near misses, and identity controls by project; the right panel shows the frozen train, development, and held-out allocation. An em dash means that no accepted row exists in a cell; explicit exclusions remain outside the denominator. Counts are deterministic static provenance diagnostics with no inferential interval, and no emitted object is executed. The figure does not estimate model accuracy or prove unrestricted semantic equivalence.

under `initial_rax=0`, the near-miss pair has equal terminal output, while another declared intervention separates it. The result is recorded in `../output/data/theorem_or_countertheorem.json` with promotion status `fail_close` `d_no_broad_theorem`. This is a useful scientific result: the intervention family has separating power only when it contains states that activate the relevant semantic difference. The single-state claim is falsified, not softened into a theorem-shaped narrative.

The receipt separates 1 kernel-checked theorem group(s) from 3 computationally validated finite-policy result(s); broad trace-refinement, compiler-transfer, and independent-replay claims remain unresolved. No broader theorem or complete x86-64 claim is promoted.

Theorem-or-countertheorem status map

Formal and computational outcomes are separated by status; falsified and unresolved candidates remain visible, and no theorem-shaped artifact is promoted without its checked boundary.

status	candidate result	assumptions / artifact scope
proved	Word64 modular accumulator identities in the declared Lean fragment	output/data/formal_verification.json
computational	Stepwise trace refinement for the transparent fixture receipt	all declared steps in repository-owned fixtures only
unresolved	Observation projection preserves the declared typed fields	a general Lean theorem connecting the Python observation encoder remains open
computational	The finite source-bound quotient relation is an equivalence relation	finite generated catalog and declared intervention set only
computational	Exact optimality of the finite sensing policy	finite declared sensor/action space
falsified	The single intervention <code>initial_rax=0</code> separates every declared near-miss family in the source-bound quotient benchmark.	scope not recorded
unresolved	Compiler/source/architecture transfer guarantee	no translation-validation theorem is promoted from static capability rows

proved: 1
computational: 3
empirical: 0
falsified: 1
unresolved: 2

shared assumptions: bounded transparent fixture bytes; deterministic Unicorn execution; declared initial-state intervention catalog; terminal RAX relation as the quotient observation

Figure 59: The theorem-status map contains 7 candidate results: 1 proved, 3 computationally validated, 0 empirically supported, 1 falsified by counterexample, and 2 unresolved. Each row retains its assumptions, state and memory bounds, supported instruction scope, observation/intervention contract, and proof or counterexample artifact. Kernel-checked Lean results are separated from Python/Capstone/Unicorn computational checks, while falsified and open candidates remain visible. The figure therefore controls claim promotion: a theorem-shaped file is not itself a theorem, and a bounded computational result is not a broad mathematical guarantee.

The status map (fig. 59) is intentionally a promotion ledger rather than a list of aspirational theorem titles. A **proved** row is backed by the checked formal artifact and its assumptions; a computationally validated row is bounded by the enumerated state space; a falsified row retains its smallest counterexample; and an unresolved row is not converted into a theorem by prose. This separation lets the manuscript report productive negative results without confusing them with either proof or failure of the unrestricted scientific question.

4.11 Formal counterexample search

The theorem phase above resolves one deliberately strong intervention claim. To make that falsification discipline routine rather than episodic, a smallest-first exhaustive search resolves a declared list of 8 candidate theorems against the same bounded reference interpreter used for the stepwise correspondence receipt (`../output/data/formal_trace_correspondence.json`). Each candidate declares its finite enumeration domain in advance; the search enumerates inputs in increasing declared size order until a witness fails the property or the domain is exhausted, and every falsified witness is re-verified by an independent second interpretation of the same reference semantics before it is recorded. The receipt is `../output/data/counterexample_search.json`, with declared method `deterministic`

smallest-first: operand values ascending from 0 through the declared prefix length, then the 64-bit boundary extremes $\{2^{63}-1, 2^{63}, 2^{64}-1\}$ ascending; tuples lexicographic in declared operand order.

The search falsifies 4 candidate(s) and resolves 4 as holding across every enumerated input. The first declared falsified candidate is `sub_equals_add_of_negation_incl_cf` (witness rank 2). Both outcome kinds are retained as evidence: a falsified candidate narrows the plausible claim, while a holds result is valid only inside its declared search domain. The figure (fig. 60) reports each candidate’s searched-domain size and outcome; domain sizes are exact enumeration counts, not samples, so no population uncertainty is claimed.

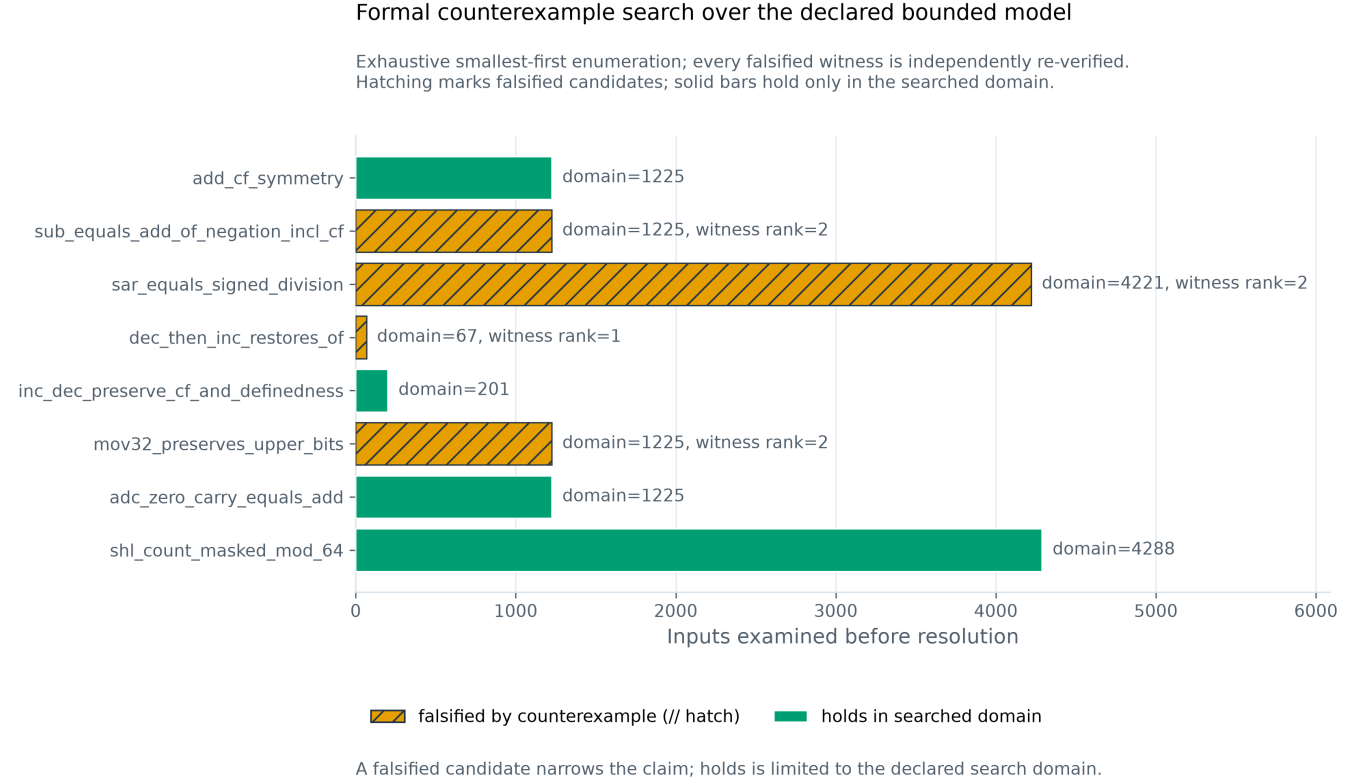


Figure 60: Smallest-first exhaustive counterexample search over the declared bounded x86-64 reference model. Each bar is one declared candidate theorem resolved inside its declared finite domain; orange bars were falsified by a verified minimal witness and green bars held across every enumerated input. Witness ranks are re-checked by an independent second interpretation of the same reference semantics. Falsified candidates remain part of the evidence and narrow the claim; a holds result is bounded to the searched domain and establishes no complete-ISA theorem.

The search strengthens the bounded instrument only. It does not prove a cross-language theorem, establish complete x86-64 semantics, or promote a holds-in-domain row to an unrestricted identity; those remain behind the promotion gates recorded in the limits and roadmap section.

4.12 Active-sensing stress study

The finite sensing certificate is exact, but exactness on one grid is not robustness evidence. The active-sensing stress study therefore re-evaluates the declared label-blind typed-field sensor channels over 240 horizon, budget, policy, and cost-weight combinations. Every row is scored against an exact finite-horizon dynamic-programming reference that is itself cross-checked against the repository’s reference planner; the largest regret of any policy rollout on this grid is 0.454 in objective units, and each row records whether its uncertainty class is exact enumeration or a fixed-seed rollout.

Budgets change the answer, not just the cost. The budget-constrained value curve reports 3 action budgets against the unconstrained exact bound; the unattained gap that a finite budget forgoes is plotted as a gap and is never zero-filled. Separately, the local timing calibration is repeated and the receipt records the maximum relative coefficient drift

31.471 across runs; these timing rows are machine-specific measurements on the calibration host, not a cross-machine cost law. The stress grid is visualized in fig. 61.

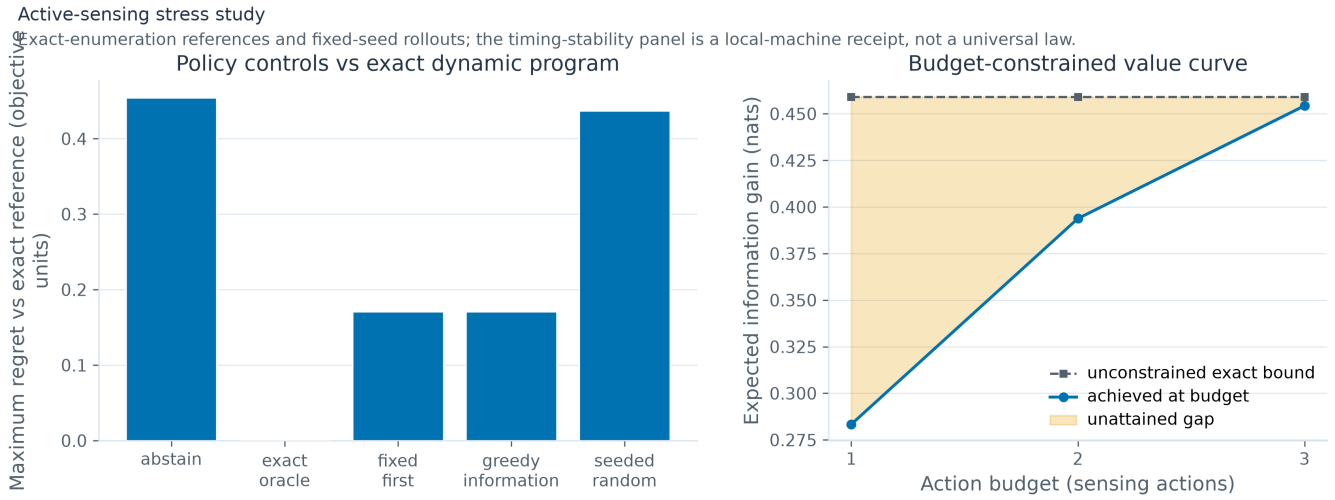


Figure 61: Active-sensing stress study over the declared label-blind typed-field sensor channels. The left panel shows each policy control’s maximum regret against the exact finite-horizon dynamic-programming reference across the horizon, budget, and cost-weight grid; the right panel shows the budget-constrained value curve with the exact unconstrained bound and the unattained gap that finite budgets forgo. Regret and gaps are exact-enumeration quantities; the accompanying calibration-stability rows repeat the local timing regression and are machine-specific measurements, not a universal information-per-cost law.

The study improves the finite robustness evidence for the sensing tranche. It does not establish a universal information-per-cost law, cross-machine calibration invariance, or semantic discovery; those boundaries remain explicit in the receipt and in the limits section.

5 Discussion

The bridge is rigorous in a deliberately local sense: the observation vocabulary, likelihood matrix, prior, package provenance, trace hash, assembly listing, and update rule are explicit and regenerated from the same artifacts. The assembly listing keeps instruction bytes, decoded operands, dynamic addresses, and categorical labels visibly distinct. That distinction matters because the ISA reference defines architectural effects, whereas a mnemonic string alone is not a proof of dataflow, control-flow intent, or semantic equivalence [Intel Corporation, 2026]. The project consequently treats Capstone and Unicorn as evidence-producing mechanisms for a bounded fixture, not as an oracle for program understanding.

The limitation is equally explicit. The canonical phase fixture is small, its vocabulary is hand-authored, the headline Active Inference comparison uses fixed likelihood and transition parameters, and steps within a trace are dependent. The quotient benchmark with disjoint family identifiers increases the number of synthetic families and tests a co-primary model portfolio, but its held-out scores remain bounded benchmark endpoints rather than evidence of population-level semantic identifiability. The supervised leave-one-fixture-out fit estimates tensors without using the held-out fixture, yet its small training complements remain a generalization diagnostic. The normalized state now makes flags, widths, memory bounds, stack effects, and unsupported instructions explicit; it does not resolve calling conventions, unbounded aliasing, indirect targets, privilege state, exceptions, or architecture-specific side effects. Binary-analysis platforms such as BAP and GTIRB show why intermediate representations and tool interchange become important as analyses scale [Brumley et al., 2011, Schulte et al., 2019]. Future modules can add those features through new checked adapters, but each extension should enter with its own semantics, tests, provenance, and claim boundary.

The quantitative results are honest and explanatory rather than inflated. On this corpus the temporal HMM beats the identity-transition filter but not the memoryless per-step decoder — a non-result that the ablation turns into a mechanism: because the authored likelihood is near-diagonal, a single observation already reveals the phase, so transition structure adds little. Sweeping the likelihood toward uniform, the temporal decoder (and the smoothed temporal segmentation) is exactly where the advantage appears: per-step decoding collapses while the transition structure carries the information (main manuscript fig. 48). Phase segmentation is now measured against the hand-authored ground truth (main manuscript fig. 49), and both the boundary-recovery result and the mechanism are registered as bounded claims (CLM-014, CLM-015). This framing keeps the headline faithful: no net temporal advantage at an informative likelihood, a clear one under ambiguity, and every number recomputed by the validation oracle rather than asserted.

The robustness machinery added around the quotient portfolio changes how the endpoint is defended, not what it claims. Hyperparameter selection is development-only, five deterministic starts are rerun and their seed ranges are recorded separately from the source-family clustered intervals, and Platt calibration is reported alongside — the uncalibrated balanced accuracy and macro-F1 stay primary and the calibrated metrics gate no claim. The family-shuffle control, the bounded metadata probe (held to its own boundary: bounded probe only; no architecture-metadata claim), the declared near-duplicate template siblings, and the cross-encoding consistency check all qualify the endpoint rather than strengthen it. No superiority claim is gated by any of this machinery: the retrospective correction withholds all such claims, and the recurrent model’s at-or-below-chance exclusion from the within-family label-scramble diagnostic is declared in the receipt rather than absorbed into a favorable summary. On the temporal side, the optimization-rewrite fixture plays the analogous negative role: its `lea` rewrite shares the residual `other` observation category while its latent phase is accumulator update, and that designed ambiguity is retained as measured evidence that the corpus can hold cases the decoder does not resolve.

This is the intended relationship to the Free Energy Principle [Parr et al., 2022]: assembly traces become sensory evidence for a generative model whose beliefs are updated under an explicit variational objective. The actual GNN parser supplies the admitted model tensors, pymdp executes both the comparison diagnostic and prospective sensor Agent, and RxInfer supplies separately scoped native Bethe free-energy diagnostics [Friedman, 2026, Heins et al., 2022, Bagaev et al., 2023]. The sensor experiment establishes a causal acquisition order: policy selection precedes the chosen Capstone or Unicorn observation. It does not establish that the authored fixture hypotheses are discovered semantics or that the model-derived utility is a measured physical cost. The source-bound external track adds a complementary validity surface: pinned permissively licensed revisions are compiled and decoded locally, source-contract edits are recorded, and emitted objects remain outside execution. Its accepted rows establish a documented provenance denominator and a project-held-out allocation; they do not turn lexical witnesses into universal equivalence proofs or static coverage into application effectiveness. The model is a reproducible computational substrate for asking which temporal and active-inference mechanisms are visible once assembly evidence, formal scope, and external provenance

have all been made explicit.

Requiring GNN sources for the fitted and discriminative comparators makes the parameter boundary inspectable across that substrate. The generated catalog binds numeric instances, training context, adapter code and rendered topology. It supports a reproducibility claim: the recorded parameters are the parameters admitted to inference. It does not show that a chosen architecture is adequate for program semantics. The categorical energy diagnostic illustrates the same distinction. A stationary iteration trace is expected for the exact fixed chain; prefix evidence reveals how each observation changes the model’s evidence, while supplying no learning curve or general convergence guarantee [RxInfer contributors, 2026a,b].

The additional formal and cross-architecture tranches clarify what a stronger scientific contribution would require. The Lean receipt gives the quotient program a genuine kernel-checked algebraic nucleus: equivalence, neutrality, and a separating subtraction witness are stated over every formal 64-bit initial word. That nucleus is intentionally not conflated with the Python adapter. A theorem relating the Lean small-step semantics to the Capstone and Unicorn trace rows remains a separate obligation, because agreement between two executable paths is not by itself a proof of correspondence. Similarly, the three-ISA fixture benchmark demonstrates that real decoder/emulator APIs can be exercised on a shared numeric toy operation, but it does not compare flags, memory, exceptions, ABIs, compiler transformations, or semantic quotients [Lean Community, 2026, Dasgupta et al., 2019]. These are useful falsification surfaces precisely because their boundaries are narrower than the claims they might tempt a reader to make [Capstone Engine Project, 2026, Unicorn Engine Project, 2026].

The exact information frontier is another deliberate separation. Dynamic programming removes Monte Carlo noise from the finite sensor mechanism and shows how additional non-repeating typed-field queries trade proxy cost against expected entropy reduction. It does not turn normalized units into acquisition time or convert a model-derived channel into a physical sensor. A future result would need measured costs, source-family-clustered uncertainty, alternative sensor models, strong fixed/random baselines, and a theorem or counterexample for the proposed bound [Friston et al., 2017, Parr et al., 2022]. Thus the current advance is methodological: it makes the next conjecture executable and falsifiable without reporting the local frontier as a general law.

The replication boundary follows the same logic. The public bundle is label-blind, hash-bound, and replayable without network access; its protected scorer can compare predictions without distributing the gold labels. No independent replay has been observed yet: the recorded status is `not_yet_observed` in the `not_observed` control scope, while the replay interface and the protected-scoring endpoint remain frozen and ready. A recorded controlled replay would demonstrate that the frozen public interface crossed a distinct control boundary; it would not be an independent physical-machine run, an independent reimplement, or independent scientific review, and it would provide no external confirmation of the inference claim. Keeping these evidence states separate prevents an engineering rerun from being mistaken for scientific generalization.

The new preparation-audit receipt extends the same discipline to the external-corpus path without changing its claim boundary. Its eight steps — manifest parsing, archive digest verification, archive-safety rehearsal, license checking, static compilation, build reproducibility, label-contract conformance, and exclusion accounting — are validated as a rehearsal-only preparation path. The receipt is not an efficacy claim: it performs no execution, P13 licensed external validation remains underpowered, and promotion still requires MAJ-01, an independent replay of the acquisition/build recipe. The MED-05 separately controlled replay boundary is likewise unchanged and locally blocked by design (`blocked_external`); no local rerun, however complete, substitutes for it.

5.1 What the visual atlas adds to the scientific argument

The visual atlas is not an ornamental supplement. It is a second, inspectable projection of the evidence graph. The architecture and state plates show where static decoding ends and dynamic state begins; the quotient construction and statistical hierarchy show why source roles, interventions, and clusters cannot be collapsed into one row count; the coverage matrix and theorem-status map show which formal statements are checked and which remain outside the contract; and the replay audit shows the protected failure boundary rather than hiding it in a log file. This is especially important for assembly research, where a readable listing can create an impression of semantic completeness even when the actual observation channel is narrow.

The atlas also makes absence legible. An unavailable compiler target is not a zero-valued performance cell; an unsupported instruction is not a failed semantic transition; a missing observation is not an observed zero; and a one-run controlled replay is not an independent sample. Text labels, hatching, direct annotations, and accessible tables reinforce those distinctions without requiring color perception. The long descriptions and source hashes provide a

route from a visual claim back to the producer receipt, while the visible captions state the population and uncertainty boundary in the place where a reader is most likely to encounter the result.

The added observation-semantics plate is useful because it locates these negative spaces before any model score is interpreted. It treats the typed record as an evidence object with a vocabulary, not as a bag of numerical defaults: `__MISSING__` means that an observation was not supplied, whereas `__UNKNOWN__` means that a categorical value was outside the fitted support. Unsupported and unavailable statuses describe contract boundaries, while a zero is retained only when a receipt measured zero. The public/private replay boundary is shown in the same vocabulary, so protected labels remain a scoring resource rather than an input feature. This is an important methodological discipline, but it does not by itself prove that the observation channel is complete, that a protected boundary has no side channel, or that the finite quotient relation is unrestricted program equivalence.

The central interpretation therefore remains modest but useful. The package demonstrates how to make an assembly-to-inference experiment inspectable across representation, execution, formal, statistical, and document boundaries. It does not demonstrate that a model has recovered program meaning in the unrestricted sense. Its scientific value lies in making the stronger claim harder to assert without the corresponding source-bound evidence, and in turning the next stronger claim into an explicit, falsifiable research task.

6 Validation and Scholarship Boundary

The project writes a source ledger with 52 entries to `../output/data/scholarship_sources.json`. Each source is assigned a project role and a claim boundary, so software-package documentation, peer-reviewed active inference theory, and optional binary-analysis adapters do not collapse into a single undifferentiated support claim. The role partition is shown in fig. 62. These counts describe the curated reading inventory. Multiple sources may document the same implementation or motivate the same method; they are not independent replications or quantitative weights of support. The stage totals in fig. 63 retain empty categories explicitly. Bibliography and citation closure establish traceability, while the adjacent source-bound computations must establish each project result.

The project also writes a claim register with 40 bounded claims to `../output/data/claim_register.json`; 40 are linked to sources, generated artifacts, validation checks, and explicit boundary text. The linked evidence matrix and claim-to-evidence map are shown in fig. 63 and fig. 64. The generated artifact surface is separately recorded in `../output/data/reproducibility_manifest.json`, which tracks 119 analysis artifacts by file size and SHA-256 digest; the manifest is visualized in fig. 65.

The sibling docxology/template renderer is also configured to emit a standard combined DOCX at `../output/docx/active_assembly_theory_and_practice_combined.docx`. Its post-render receipt at `../output/data/docx_renderer_receipt.json` records the observed document status, embedded-image count and caption count. It checks OOXML ZIP integrity, required document parts, figure/caption parity, unresolved tokens, the hydrated manuscript hash, and the substantive figure-registry hash. The two validator-produced summary figures are excluded from that digest because their own checks are self-referential. This is a format and freshness contract; it does not establish PDF/UA tagging, reading order, or scholarly validity, which remain separate gates.

The generated cover illustration is source-controlled at `manuscript/assets/active_assembly_cover.png` and is included through the template’s config-driven paper-cover contract. The same validation surface checks the bounded symbolic semantics report, stepwise transfer-function/Unicorn refinement, direct-target CFG partition, held-out fitting membership, and cover-image presence/nonblankness; these checks are engineering and provenance contracts, not evidence of scientific generality.

The shared artifact boundary is separately audited rather than inferred from the reproducibility manifest. The current receipt layer is schema 2.0, with 10 required common fields across 51 project-owned JSON artifacts and 0 envelope errors. The audit command is `uv run python scripts/run_scientific_program.py audit-artifacts`. Domain schema versions remain intact; the receipt layer adds protocol, producer, input, toolchain, limitation, and status lineage, visualized in fig. 66. This design follows provenance and research-object packaging practice [Lebo et al., 2013, Sefton et al., 2023, Sandve et al., 2013], while preserving the important distinction between local reproducibility and independent replication [Association for Computing Machinery, 2020].

The analysis validator reports 195 passing checks out of 195, with 0 failures and `all_passed = true`. The report is written to `../output/data/validation_report.json` and summarized in fig. 67. It checks trace schema fields, sample hashes, required package provenance, trace observation semantics, dynamic-register trace shape, RxInfer free-energy finiteness, RxInfer companion artifact consistency, categorical-companion model identity, memory-operand to dynamic-event joins, data-hash consistency, reproducibility-manifest hashes and byte sizes, figure nonblankness, source-ledger bibliography coverage, manuscript citation usage, citation closure, source-shape quality, source-stage and evidence-tier partitioning, claim-register links, fixed-parameter generative adequacy, and cautious claim-boundary language. The newer check families added with the landed tranches are contract-plus-freshness pairs for the compiler matrix (`compiler_transfer_contract/compiler_transfer_freshness`), the quotient classifier and its invariance protocol (`quotient_classifier_contract/quotient_classifier_freshness` and `quotient_invariance_contract/quotient_invariance_freshness`, which together pin the development-only hyperparameter selection, deterministic-start ledger, family-shuffle control, bounded metadata probe, near-duplicate declarations, and cross-encoding consistency), and the external preparation audit (`external_preparation_audit_contract/external_preparation_audit_freshness`).

This validation layer is intentionally adversarial. It is designed to fail if a future edit reports RxInfer free energy without finite JSONL values, adds a source-ledger entry without a bibliography record, drops a required adapter provenance row, changes observation labels without updating category IDs, drops RxInfer posterior/configuration artifacts, changes a manifest-tracked artifact after hash recording, bypasses the common artifact envelope, adds an unlinked claim, or renders empty figures while the basic pipeline still exits successfully.

Counts describe curation roles;
source totals do not measure empirical support.

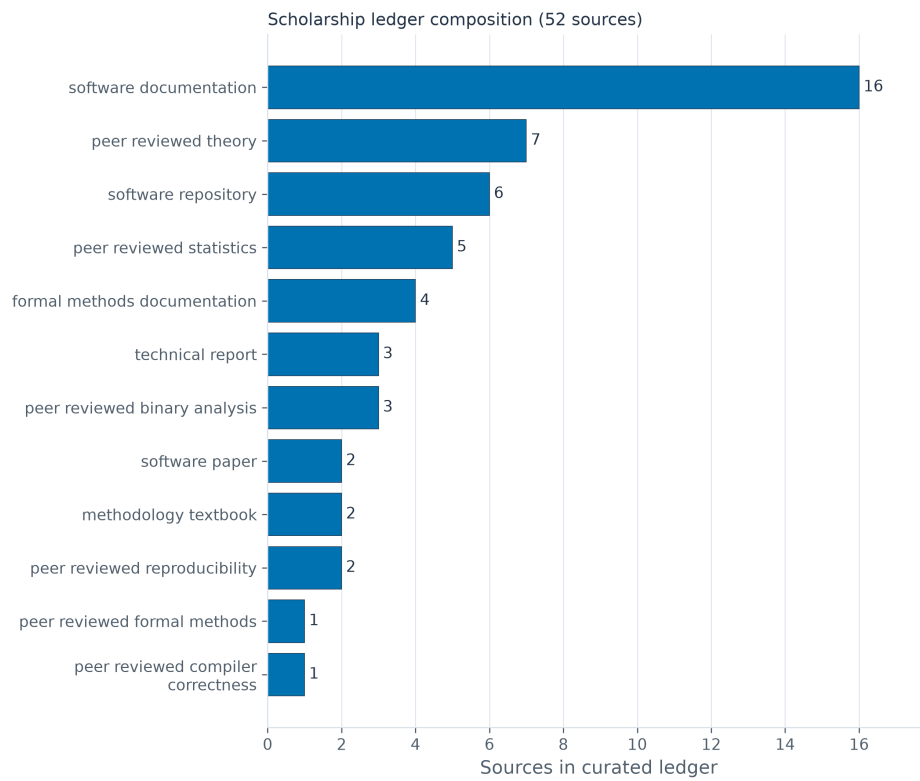


Figure 62: Counts of sources in the curated scholarship ledger by source type. The ledger separates peer-reviewed theory and binary-analysis context from software papers, package documentation, and repository-level roadmap references; counts describe support roles, not evidence strength by themselves.

Each source contributes to one cell; zero means no ledger entry.
Source counts describe roles, not independent experimental evidence.

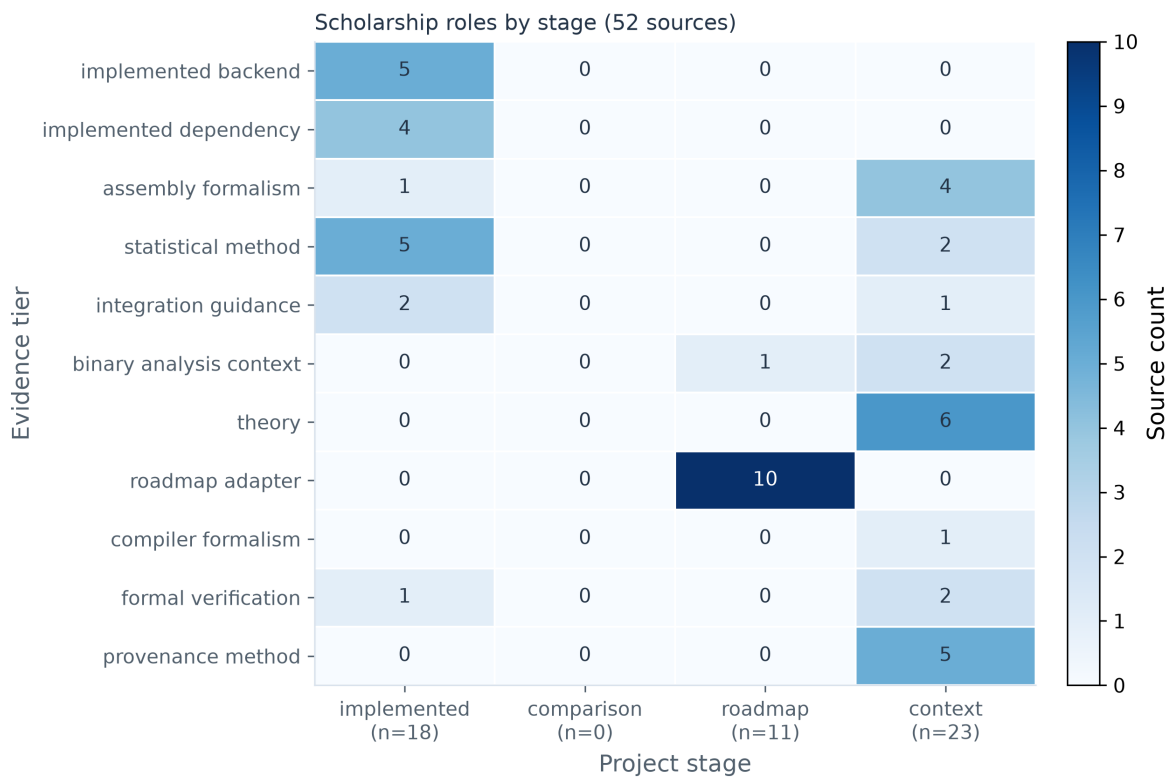


Figure 63: Scholarship-source counts crossed by project stage and evidence tier. Implemented and comparison rows identify sources used to contextualize executed paths; roadmap and context rows remain explicitly outside the currently executed evidence-producing surface.

Each claim row has three vertically separated evidence markers; counts audit traceability, not independent replication.

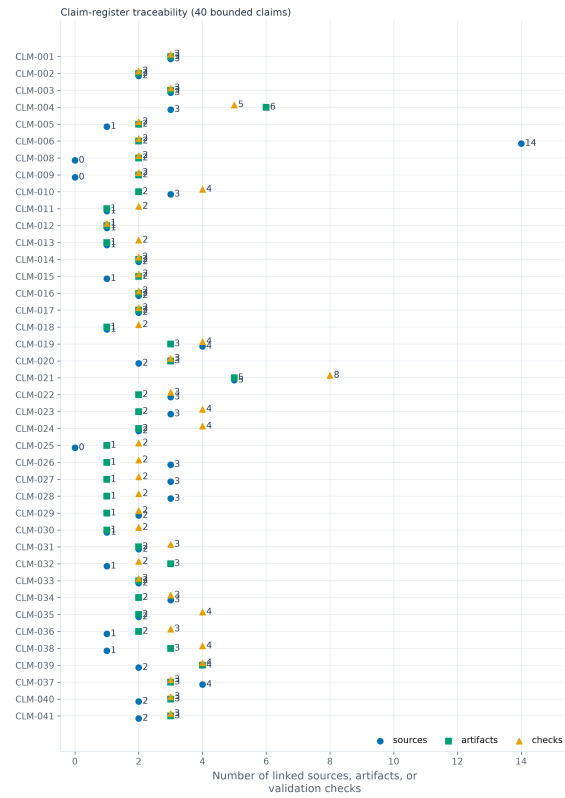


Figure 64: Claim-register coverage map. For each bounded claim, vertically separated markers count linked source keys, generated artifacts, and validation checks. These counts audit traceability; they do not turn a linked claim into independent scientific replication.

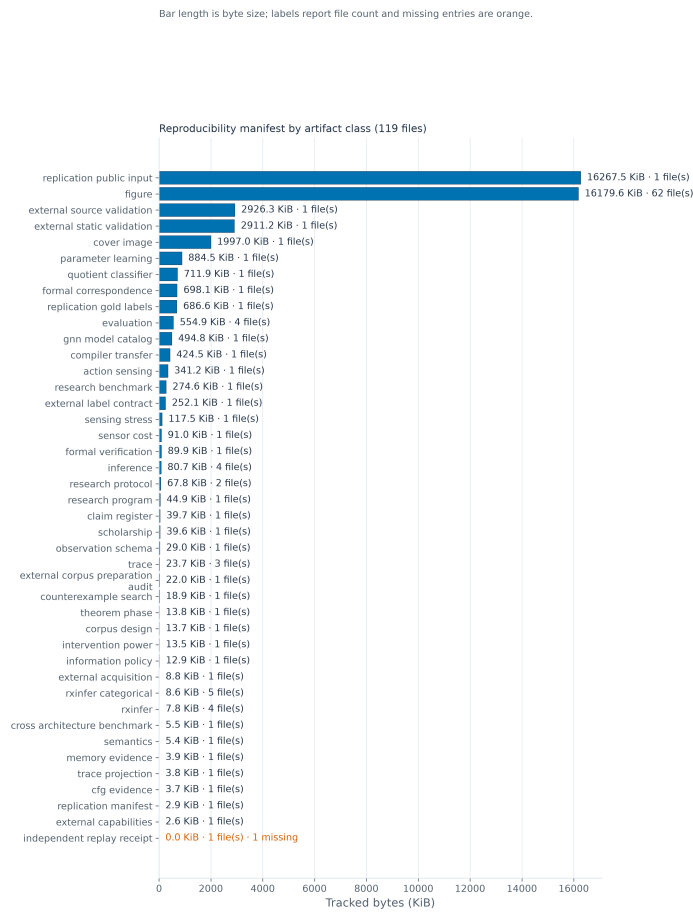


Figure 65: Reproducibility-manifest coverage by artifact class. Bar height is the total tracked file size in KiB and the annotation gives the number of files; the manifest independently records each file’s SHA-256 and byte size so stale or replaced artifacts fail validation.

Protocol-v2 artifact receipt closure

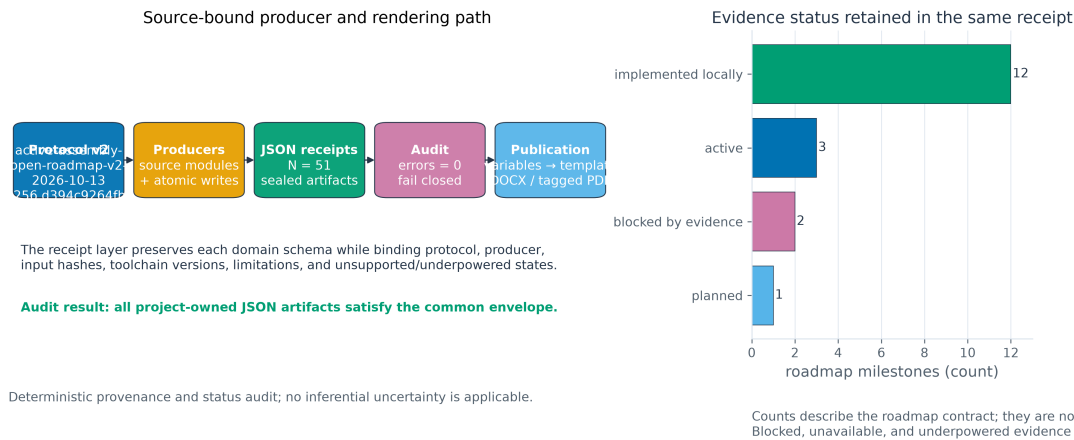


Figure 66: Protocol-v2 artifact receipt closure for 51 project-owned JSON artifacts. The flow binds the canonical protocol to producer modules, atomically written domain payloads, surface validation, and the hydrated manuscript/DOCX/tagged-PDF boundary; the companion bars retain the 18-milestone evidence-status contract. The audit found 0 receipt error(s). Counts are deterministic provenance checks with no inferential uncertainty; blocked, unavailable, and underpowered evidence remains labelled rather than rendered as zero performance, and this figure does not establish scientific promotion [Lebo et al., 2013, Sefton et al., 2023, Sandve et al., 2013, Association for Computing Machinery, 2020].

Compact domain view; individual check names and details remain in validation_report.json.

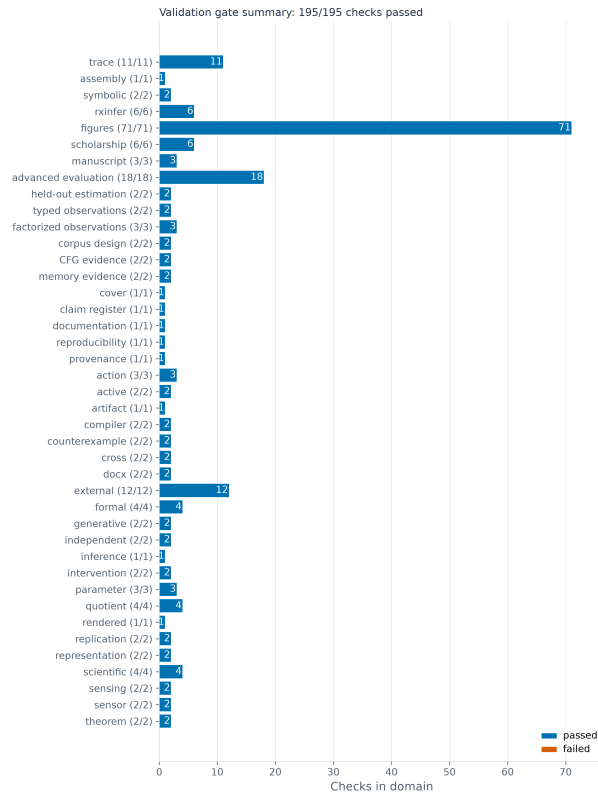


Figure 67: Generated validation checks for trace structure, assembly-listing lineage, inference artifacts, figures, scholarship, manuscript hydration, provenance, and reproducibility. A pass means the specific project oracle evaluated true; it is not a substitute for scientific peer review.

6.1 Open-roadmap receipt gates

The extended gates are source-bound rather than green-by-construction. The protocol hash is recomputed before validating the quotient, correspondence, compiler, sensor, replication, and theorem-phase receipts. Each producer is then rerun and compared with the disk artifact, with the timing-only part of the sensor receipt excluded from the deterministic freshness comparison because wall-clock calibration is expected to vary. Contract checks separately require label blindness, three-way family splits, nonempty metrics, explicit unavailable states, static-only compiler analysis, exact public/gold separation, and a proved-or-falsified theorem boundary.

The new source-bound artifact family is:

6.1.1 Model, execution and sensing receipts

Surface	Contract that is actually checked	Promotion boundary
quotient classifier	protocol hash, label-blind features, disjoint training/development/held-out families, finite metrics and clustered intervals; development-only hyperparameter selection, five-start ledger, family-shuffle and bounded metadata-probe controls, declared near-duplicate siblings, and cross-encoding consistency	bounded endpoint only; semantic discovery is not established and the retrospective correction withholds superiority claims
formal correspondence	real Capstone decode, bounded Unicorn trace, Python reference step alignment, modeled flags, and Lean prerequisite	declared transparent subset only; no complete x86-64 theorem
compiler transfer	exact local target command, source/object/disassembly hashes, Capstone decode, no object execution, explicit unavailable cells	capability receipt only; no compiler-generalization or application claim
sensing costs	deterministic vectors, machine-specific calibration, exact finite dynamic-program objective and gap	finite benchmark certificate only; no universal optimality law

6.1.2 External source and label contracts

Surface	Contract that is actually checked	Promotion boundary
external static corpus	10 pinned source rows, archive/license digests, static compilation, disassembly hashes, Capstone coverage, and explicit failed rows	static coverage and lineage only; no application effectiveness
source-bound label contract	100 accepted rows across 5 projects and two revisions per project: 40 lexical-equivalence witnesses, 40 contract-directed near misses, and 20 identity negative controls; base/pair source, object, and disassembly hashes; project-held-out test split	source-contract and static provenance evidence only; lexical comments are not universal equivalence proofs, near misses are not executed behavioral labels, and no application claim follows

6.1.3 Replay, formal status and acquisition gates

Surface	Contract that is actually checked	Promotion boundary
replication bundle	public/gold field separation, public protocol/toolchain/instruction hashes, identifier/hash equality, no-network replay metadata	frozen interface and local replay contract; no independent implementation claim
replay ingestion	protected scoring boundary, explicit separate-control attestation, mismatch retention, and source-bound receipt hashes	<code>not_yet_observed</code> in <code>not_observed</code> ; observed controlled replay does not imply independent physical-machine replication or independent reimplementation
theorem phase	checked Lean theorem list plus generated bounded counterexample	fail-closed; unresolved claims remain unresolved
external preparation audit	eight preparation steps (manifest parse, archive digest verify, archive-safety rehearsal, license check, static compile, build reproducibility, label contract, exclusion accounting) each validated without execution or network use	rehearsal-only preparation path; not an efficacy claim; promotion requires MAJ-01 independent replay of the acquisition/build recipe

The registry and figure validator apply the same source-bound rule to the visualization suite. A caption is not accepted merely because it exists: its input artifacts, claim IDs, uncertainty method, sample hierarchy, units, and alt text are generated from the figure registry. Missing, unseen, unsupported, and unavailable cells must be described rather than drawn as zero. The compiler, cost, correspondence, quotient, and replication panels are therefore part of the analysis validation surface, not decorative post-processing.

7 Limits, validity, and scientific roadmap

The evidence supports a narrow methodological conclusion. A real assembly trace can be decomposed into auditable static and bounded dynamic records, projected into an explicit categorical observation channel, and evaluated with generative and discriminative diagnostics whose assumptions are visible. The package does not establish that the latent labels are intrinsic program semantics. They are hand-authored phase hypotheses used to make the inference contract measurable.

The most important validity boundary is between a decoder and a generative model. A decoder can score well on a short labelled sequence because an individual observation is highly informative. A generative model must also produce plausible state persistence, transition structure, and observation frequencies. The fixed-parameter generative adequacy analysis performs that second check by simulating 256 same-length sequences per fixture. It tests 80 scalar structural metrics; 76 fall inside the central simulation intervals, for an observed coverage fraction of 0.950. These are descriptive diagnostics of the declared A/B/D model, not posterior uncertainty intervals or evidence of population-level fit [Gelman et al., 2013]. The complete per-fixture record is available in `./output/data/generative_adequacy.json` and visualized in fig. 68. This fixed-parameter categorical phase-HMM does not establish learned semantic representations or external validity.

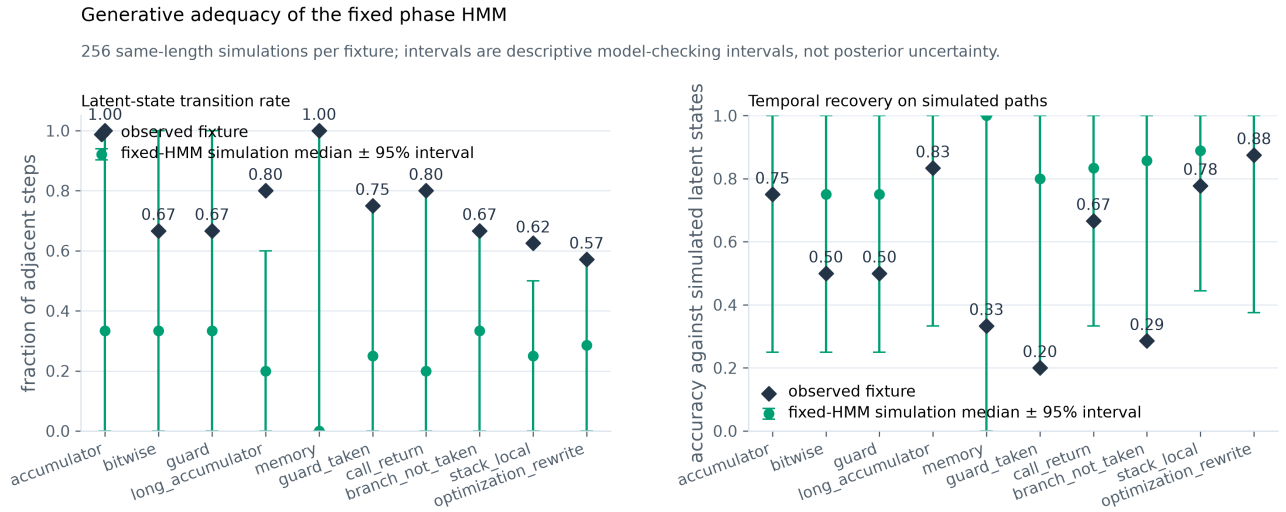


Figure 68: Fixed-parameter generative adequacy check. Diamonds show the observed phase-transition rate and temporal recovery accuracy for each fixture; circles and intervals show the median and central 95% simulation interval from same-length sequences generated by the fixed A/B/D HMM. Intervals are descriptive model-checking intervals, not posterior or population uncertainty; a mismatch identifies a concrete model/data discrepancy for the next iteration [Gelman et al., 2013].

Several limitations remain substantive. The corpus contains 10 transparent x86-64 programs — the original six register-only programs plus four designed fixtures (in-span call/ret roundtrip, not-taken guard pairs, stack-local state, and the add/inc/lea optimization-rewrite motif) — so complete fixtures rather than independent instruction steps are the meaningful sampling units. The coverage inventory records one bounded stack-memory fixture, repeated authored phases, and one taken-branch path that leaves a statically decoded instruction unexecuted; the temporal family split (4/2/4 training/development/held_out) was frozen before scoring, but these additions improve design variation without creating independent binary samples. The labels cover only five broad phases and are assigned from the known fixture construction. The observation map collapses flags, effective-address details, calling conventions, indirect targets, and unsupported effects. The bounded symbolic interpreter therefore explains only the documented transfer-function subset; agreement with Unicorn on those fixtures is a cross-check, not a complete-ISA certification. The supervised leave-one-fixture-out fit is a useful leakage diagnostic because the excluded fixture does not contribute to its fitted tensors. It is not unsupervised phase discovery, and 10 held-out evaluations cannot establish generalization to arbitrary binaries. The likelihood and transition ablations are controlled interventions on the declared model matrices. They explain the mechanism by which temporal structure matters under ambiguity; they do not measure a real-world noise distribution.

The observation-only ordered-EM tranche makes the next limitation measurable. It excludes phase labels from fitting

and reports a complete training complement for every holdout, but it still inherits a reference initialization, an ordered self/next-state scaffold, and a positive transition-support prior. Its held-out mean is 0.480, with a mean difference of -0.093 relative to the fixed reference. That result is an implementation and leakage-boundary advancement, not evidence that latent phase meanings have been discovered. Future work must compare multiple state-order constraints and deterministic initializations, report training-versus-held-out evidence separately, and add longer programs with ambiguous repeated phases before making any claim about unsupervised semantics.

The typed observation expansion addresses a different limitation: the original six-category channel was lossy by construction. The new schema retains 8 fields in 4 evidence families, and the factorized/hierarchical comparison is fit with the same fixture-level exclusion boundary. Its mean full-factorized accuracy is 0.520 versus 0.667 for the collapsed reference; the difference is -0.147. Thus the expanded typed result is a measured negative companion-model result, not a null that can be rounded into a success claim: the current composite score is lower than the collapsed reference on these held-out fixtures. The masked-memory check changes the mean to 0.562, which is 0.042 relative to the full typed score. These results are intentionally modest: most labelled fixtures execute every instruction and contain no memory operand, so the new fields are not yet independent sources of variation. The typed model is a supervised composite likelihood diagnostic, not a semantic representation, and the hierarchical score is not assumed to be a normalized joint probability.

The quotient robustness machinery has separate evidential limits. Hyperparameter selection and Platt calibration are development-only, and the uncalibrated metrics remain primary. The within-family label scramble refits on scrambled held-out labels and evaluates those same rows, preserving family class totals. Its score (0.597) is an in-sample perturbation diagnostic; it does not establish absence of family leakage. Models at or below chance are excluded from its comparator set explicitly. The encoding/opcode-surface probe uses bounded refits, with margins +0.004, +0.024, and +0.031 for the three models; it supplies no architecture-metadata conclusion. The similarity ledger retains 128 declared template-sibling flags, and the cross-encoding contract covers 61 of 64 families. The retrospective correction withholds every superiority claim. These controls cannot certify behavior outside the declared benchmark.

The active layer has the same requirement, and the first action-conditioned tranche now makes that boundary explicit. `action_sensing.json` chooses typed field queries before revealing held-out values and compares active, fixed, seeded-random, and inadmissible-oracle policies under one normalized budget. This is a genuine sensor-choice mechanism test, but it remains a small fixture-level result: the normalized action budget is not the primary measured cost vector, local timing calibration is machine-specific, the oracle is not deployable, and no autonomous sensing claim is made.

Three further tranches make the roadmap more rigorous without inflating the claim. First, the formal receipt `../output/data/formal_verification.json` reports a locally checked modular Word64 fragment under toolchain `leanprover/lean4:v4.35.0`, with 71 named theorems and status `locally_verified`. The generated bridge also records all 16 general-purpose registers, RIP, modeled flag-definedness, bounded memory and stack bounds, step index, and running status, with kernel-checked fixture invariants for register preservation, carry preservation, step advancement, and bound preservation. This earns a bounded kernel-checked state foundation for quotient witnesses, but not a correspondence theorem between Lean and the Python/Capstone/Unicorn trace. Second, the real adapter receipt `../output/data/cross_architecture_fixtures.json` exercises 3 architectures, all with bounded result 8 and execution flag `yes`. That is an integration and adapter-drift benchmark, not cross-ISA semantic equivalence or compiler invariance. Third, the exact finite sensor analysis `../output/data/active_information_frontier.json` contains 4 channels and 16 frontier rows, with exhaustive enumeration recorded as `yes`. The companion measured-cost receipt records deterministic sensor vectors and optional local calibration; both analyses remain finite and therefore preclude a general information-per-cost law.

The research program now proceeds through coupled measurement, formal, identifiability, transfer, active-sensing, and replication expansions. The typed, observation-only, action-conditioned, formal-fragment, adapter, and finite-policy first tranches are implemented as local source-bound surfaces; the correspondence theorem, source/compiler-disjoint external benchmark, measured query costs, and independent replication remain active or unearned. Each expansion must add an artifact, tests, negative controls, uncertainty definitions, and manuscript language stating what the evidence cannot show. This keeps the project ambitious in method while preventing a larger toolchain or more elaborate figure from being mistaken for stronger scientific evidence.

7.1 Long-horizon scientific program

The machine-readable successor program is `../output/data/scientific_program.json`. Of its 18 milestones, 12 are implemented only on the local source-bound surface, 3 is active, and 1 remain unearned; these counts describe program state, not scientific success.

Next, define an executable operational equivalence relation and separating intervention family, then test quotient-state invariance under semantics-preserving representations. Freeze source-family adjusted mutual information, posterior equivalence-consistency, calibration, invariance gap, and near-miss separation before scoring; source and compiler lineages must be disjoint so a model cannot define its own semantic labels.

Promotion requires counterexamples, multiple initializations, strong baselines, and independent reimplementations. A mechanically checked theorem or decisive countertheorem is separate: benchmark performance cannot substitute for it. The conjecture, falsifiers, gates, and stop conditions live in `docs/scientific_program.md` and its generated receipt.

The first P3 prerequisite is the bounded equivalence/near-miss protocol `../output/data/quotient_invariance_protocol.json`: finite initial-state interventions run through real Capstone and bounded Unicorn, requiring equivalent pairs to agree on final **RAX** and a near miss to separate. It is an operational protocol, not evidence of latent identifiability, posterior invariance, complete-ISA equivalence, or cross-architecture transfer.

The protocol now retains a richer diagnostic surface for every intervention: terminal **RAX**/**RIP**/**RSP**/**EFLAGS** projections, stepwise **RAX** and modeled flag states, memory-event counts, observation categories, and a deterministic surface digest. These projections are deliberately reported beside the declared final-**RAX** relation. An equivalent pair may differ on representation surface or intermediate state while remaining equivalent under the declared output relation; conversely, a final-state relation cannot certify unmodeled memory, flags, exceptions, or calling-convention behavior.

Benchmark receipt `../output/data/representation_invariance_benchmark.json`: 8/16 families/pairs; primary consistency/separation 1.000/1.000; 5 surface-blind controls; negative-control separation 0.625 (3 unresolved). The accompanying benchmark plot (fig. 69) reports the same endpoints with descriptive Wilson intervals whose unit is the source-program family. The intervals make the small number of independent families visible; they are not posterior or population uncertainty, and the negative-control panel is an intervention sensitivity check rather than a competing fitted model.

7.2 Current promotion ledger for P3–P17

The open roadmap is now executable in bounded local tranches, but its promotion state remains deliberately asymmetric:

7.2.1 Benchmark, formal and replication evidence (P3–P7)

Phase	Earned local evidence	What remains unearned
P3 quotient benchmark	64 quotient-family identifiers with disjoint split membership but shared authored lineages, three-way co-primary portfolio, clustered intervals, calibration, leakage controls, and negative controls	independent-lineage generalization, independent implementation and any claim of semantic discovery
P4 formal correspondence	explicit 16-register/width-aware Python state, real Capstone/Unicorn step comparison, and a separately kernel-checked Lean fragment	broad Lean/Python correspondence theorem for the full declared user-mode subset
P5 transfer	local compiler capability receipt with exact target/source/object hashes and explicit unavailable targets	compiler-disjoint, source-disjoint, project-held-out licensed-source effectiveness validation
P6 active sensing	deterministic vectors, overhead-corrected local calibration, regression diagnostics, and exact finite dynamic-program certificates	multi-hardware acquisition model and non-oracle near-optimal policy evidence

Phase	Earned local evidence	What remains unearned
P7 replication	frozen no-network CLI, digest-ordered public rows, private gold separation, hash validation, replay metadata, and recorded replay status <code>not_yet_observed</code>	independent implementation, independent physical/CI replay, and external scientific review

7.2.2 Mechanism, robustness and transfer evidence (P8–P12)

Phase	Earned local evidence	What remains unearned
P8 theorem phase	proved small algebraic theorem group, automatically generated bounded counterexample, and a smallest-first exhaustive counterexample-search receipt over declared candidates	broad trace theorem, transfer guarantee, or field-changing general result
P9 intervention power	exhaustive finite subset enumeration for the separate four-state representation pilot, minimum separating sets, and a one-state negative control	universal identifiability, intervention completeness, or a separation law outside the frozen matrix
P10 initialization robustness	five-start observation-only holdout ledger with accuracy spread and sequence agreement	stable latent-parameter recovery or semantic discovery beyond the finite fixtures
P11 normalized machine state	all-GPR, width-aware, bounded-memory reference state with boundary tests and Lean normalized-state definitions	complete cross-language proof for the declared broad subset
P12 scaled transfer/quotient	64-family quotient receipt, source/compiler/architecture capability matrix, and protocol-bound splits	external validity or universal invariance; retrospective superiority claims remain withheld

7.2.3 External evidence and promotion gates (P13–P17)

Phase	Earned local evidence	What remains unearned
P13 licensed external validation	10 pinned rows across 5 projects and two revisions per project, static-only acquisition/compiler path, Capstone decoding, 100 source-bound contract rows, and project-held-out labeling	application-facing effectiveness until source-bound label, project-held-out, build, and external acquisition/build replay gates pass
P14 measured policy robustness	deterministic/calibrated cost receipts, finite oracle certificate, stress-grid policy regret and budget-value gaps, and repeated local calibration-stability rows	universal cost law or non-oracle near-optimality claim beyond the declared grid
P15 blind replication	public/private label separation, hashed public protocol/toolchain/instructions, protected ingestion boundary, and <code>no t_yet_observed</code> in <code>not_observed</code>	independent implementation or independent physical/CI replay; controlled replay of the frozen interface is not independent reimplementation

Phase	Earned local evidence	What remains unearned
P16 promotion hardening	theorem/countertheorem status map, source-bound claim/figure/scholarship closure, and accessible regeneration path	publication or award-level claim absent independent scientific review
P17 protocol/artifact closure	protocol-v2 envelope on the project-owned JSON surface, atomic producer writes, receipt audit, and source-bound manuscript/document closure	semantic identifiability, external validity, independent replication, or theorem promotion

The theorem/countertheorem receipt is `../output/data/theorem_or_countertheorem.json`. Its fail-closed status is `fail_closed_no_broad_theorem`; the counterexample family is `zeroing_idiom`. The correct scientific response to that counterexample is to strengthen the intervention contract and state the separating assumptions, not to call the failed single-state claim “almost true.” The local formal proof count is 1, while 3 result(s) remain computationally validated rather than kernel proved.

Two bounded falsification-and-robustness receipts extend this ledger. The counterexample search (`../output/data/counterexample_search.json`) resolves 8 declared candidates with 4 falsification(s); each holds row is bounded to its enumerated domain and is not promoted to an unrestricted identity. The sensing-stress receipt (`../output/data/sensing_stress.json`) scores policy controls against exact references on a declared grid and repeats the local timing calibration; its regret and drift rows are machine-bounded diagnostics, not cross-machine laws.

A new preparation-audit receipt (`external_corpus_preparation_audit.json`) validates the eight external-corpus preparation steps — manifest parsing, archive digest verification, archive-safety rehearsal, license checking, static compilation, build reproducibility, label-contract conformance, and exclusion accounting — as a rehearsal-only preparation path. It is not an efficacy claim: nothing is executed, P13 remains underpowered, and promotion of the external-validation phase still requires MAJ-01, an independently replayed acquisition/build recipe. The MED-05 separately controlled replay boundary is unchanged and remains locally blocked by design (`blocked_external`).

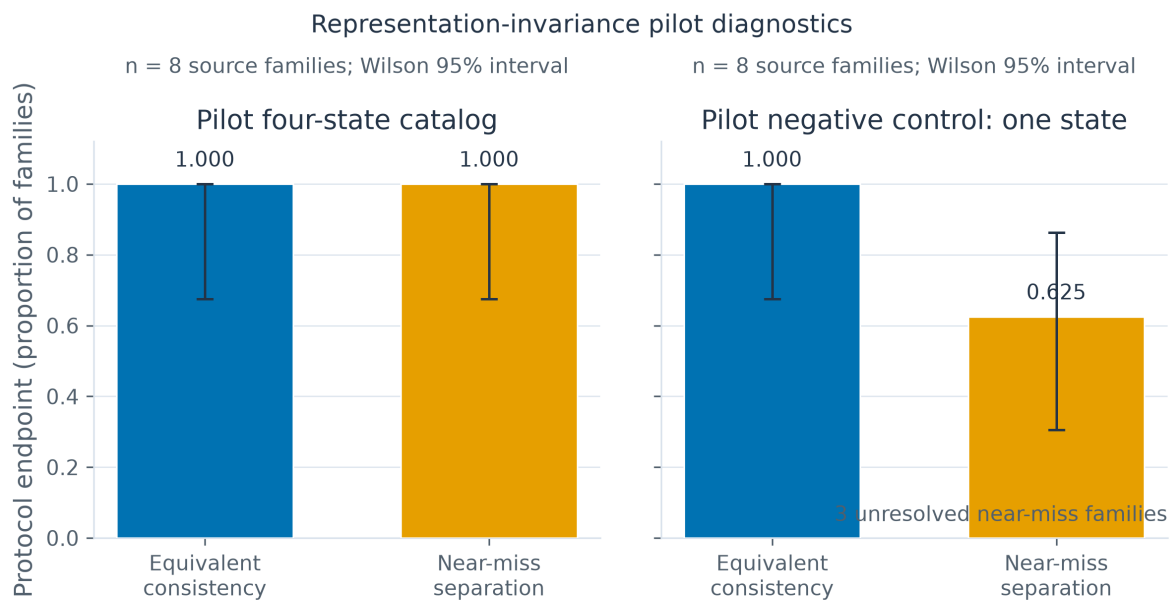
Application-facing promotion remains closed: the licensed-source receipt is `underpowered` with 5 project families and 100 labelable examples, the correspondence is `validated_bounded`, and the replication bundle reports local label separation `yes` plus a same-host controlled replay status of `not_yet_observed`. The licensed-source external gate remains `underpowered` because its independent acquisition/build replay requirement is not satisfied. Until licensed provenance, project-held-out labels, static-only safety, reproducible builds, and the external-validation replay gate pass together, this manuscript uses “demonstrator,” “bounded endpoint,” and “capability receipt” rather than “general reverse-engineering system,” “architecture invariant,” or “application effectiveness.” This rule protects scientific integrity.

7.3 Generated figure atlas

The complete visual inventory is generated alongside the manuscript rather than maintained as a second hand-edited appendix. The source-bound registry is `../output/data/figure_registry.json` and the offline accessible atlas is `../output/figure_atlas/index.html`. Together they retain all 69 substantive figures, including diagnostics that are too dense for the main narrative. Each entry records its source artifact hashes, evidence status, population grain, split, uncertainty method, visible caption, short alternative text, and long description. The cover remains a separate document asset under the existing cover-media contract.

The atlas is intentionally a scientific audit surface. It makes unavailable, unsupported, excluded, underpowered, and not-yet-observed states visible; provides exact tables for dense matrices where a chart alone would be ambiguous; and keeps deterministic enumerations separate from inferential intervals. It should therefore be read together with the claim register and the corresponding receipt, not as a collection of independent results. The canonical command is:

```
uv run python scripts/run_scientific_program.py \
  generate-figures \
  --formats png svg pdf atlas
```



Pilot intervals are family-clustered and descriptive; no fitted semantic model or population claim is shown.

Figure 69: Primary and negative-control endpoints for the separate, underpowered source-family-disjoint representation-invariance pilot. Bars report final-RAX equivalence consistency and near-miss separation over 8 source-program families; whiskers are descriptive 95% Wilson intervals computed at the family level. The single-zero-state catalog is intentionally insufficient and leaves 3 near-miss families unresolved. These are bounded Capstone/Unicorn protocol diagnostics, not fitted semantic-model performance, population uncertainty, or evidence of universal separation. The scaled 64-family co-primary classifier benchmark is reported separately.

8 References

References are rendered from `references.bib`.

- angr Project. angr: A python binary analysis platform, 2026. URL <https://angr.io/>.
- Association for Computing Machinery. Artifact review and badging - current. ACM Publications Policy, 2020. URL <https://www.acm.org/publications/policies/artifact-review-and-badging-current>.
- Avast. Retdec: A retargetable machine-code decompiler based on llvm, 2026. URL <https://github.com/avast/retdec>.
- Dmitry Bagaev, Albert Podusenko, and Bert de Vries. Rxinfer: A julia package for reactive real-time bayesian inference. *Journal of Open Source Software*, 8(84):5161, 2023. doi: 10.21105/joss.05161.
- Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001. doi: 10.1214/ss/1009213286. URL <https://projecteuclid.org/journals/statistical-science/volume-16/issue-2/Interval-Estimation-for-a-Binomial-Proportion/10.1214/ss/1009213286.pdf>.
- David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz. Bap: A binary analysis platform. In *Computer Aided Verification*, pages 463–469, 2011. doi: 10.1007/978-3-642-22110-1_37. URL https://link.springer.com/chapter/10.1007/978-3-642-22110-1_37.
- A. Colin Cameron, Jonah B. Gelbach, and Douglas L. Miller. Bootstrap-based improvements for inference with clustered errors. *The Review of Economics and Statistics*, 90(3):414–427, 2008. doi: 10.1162/rest.90.3.414. URL <https://doi.org/10.1162/rest.90.3.414>.
- Capstone Engine Project. Capstone: The ultimate disassembler, 2026. URL <https://www.capstone-engine.org/>.
- CEA SEC. Miasm reverse engineering framework, 2026. URL <https://github.com/cea-sec/miasm>.
- Sandeep Dasgupta, Daejun Park, Theodoros Kasampalis, Vikram S. Adve, and Grigore Rosu. A complete formal semantics of x86-64 user-level instruction set architecture. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1133–1148, 2019. doi: 10.1145/3314221.3314601. URL <https://pldi19.sigplan.org/details/pldi-2019-papers/24/A-Complete-Formal-Semantics-of-x86-64-User-Level-Instruction-Set-Architecture>.
- Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37. URL https://doi.org/10.1007/978-3-030-79876-5_37.
- Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. doi: 10.1111/j.2517-6161.1977.tb01600.x. URL <https://academic.oup.com/jrsssb/article/39/1/1/7027539>.
- Dyninst Project. Dyninstapi: Tools for binary instrumentation, analysis, and modification, 2026. URL <https://github.com/dyninst/dyninst>.
- Daniel Ari Friedman. Generalized Notation Notation (GNN). Software repository, 2026. URL https://github.com/ActiveInferenceInstitute/Generalized_Notation_Notation/tree/46459949944e3217f8a6a804d4ff8c52829ef275. Source revision 46459949944e3217f8a6a804d4ff8c52829ef275; DOI identifies the project concept.
- Karl Friston. The free-energy principle: a rough guide to the brain? *Trends in Cognitive Sciences*, 13(7):293–301, 2009. doi: 10.1016/j.tics.2009.04.005. URL <https://pubmed.ncbi.nlm.nih.gov/19559644/>.
- Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, and Giovanni Pezzulo. Active inference: A process theory. *Neural Computation*, 29(1):1–49, 2017. doi: 10.1162/NECO_a_00912. URL <https://pubmed.ncbi.nlm.nih.gov/27870614/>.
- Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. CRC Press, 3 edition, 2013. ISBN 9781439840955. URL <https://www.routledge.com/Bayesian-Data-Analysis/Gelman-Carlin-Stern-Dunson-Vehtari-Rubin/p/book/9781439840955>.
- Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1321–1330, 2017. URL <https://proceedings.mlr.press/v70/guo17a.html>.
- Conor Heins, Beren Millidge, Daphne Demekas, Brennan Klein, Karl Friston, Iain D. Couzin, and Alexander Tschantz. pymdp: A python library for active inference in discrete state spaces. *Journal of Open Source Software*, 7(73):4098, 2022. doi: 10.21105/joss.04098.
- Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979. doi: 10.2307/4615733. URL <https://www.jstor.org/stable/4615733>.

- Intel Corporation. Intel 64 and ia-32 architectures software developer’s manuals. Intel Architecture Documentation, 2026. URL <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>.
- Robert E. Kass and Adrian E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995. doi: 10.1080/01621459.1995.10476572. URL <https://doi.org/10.1080/01621459.1995.10476572>.
- Keystone Engine Project. Keystone engine, 2026. URL <https://www.keystone-engine.org/>.
- Lean Community. The lean language reference. Lean Documentation, 2026. URL <https://lean-lang.org/doc/reference/latest/>.
- Timothy Lebo, Satya Sahoo, and Deborah McGuinness. The prov ontology. W3C Recommendation, 2013. URL <https://www.w3.org/TR/2013/REC-prov-o-20130430/>.
- Dennis V. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4):986–1005, 1956. doi: 10.1214/aoms/1177728069. URL <https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-27/issue-4/On-a-Measure-of-the-Information-Provided-by-an-Experiment/10.1214/aoms/1177728069.full>.
- LLVM Project. Clang compiler user’s manual. Clang Documentation, 2026a. URL <https://clang.llvm.org/docs/UsersManual.html>.
- LLVM Project. llvm-objdump: Llvm’s object file dumper, 2026b. URL <https://llvm.org/docs/CommandGuide/llvm-objdump.html>.
- NASM Project. Nasm: The netwide assembler, 2026. URL <https://www.nasm.us/docs.html>.
- National Security Agency Research Directorate. Ghidra software reverse engineering framework, 2026. URL <https://github.com/NationalSecurityAgency/ghidra>.
- George C. Necula. Translation validation for an optimizing compiler. In *Proceedings of the 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 83–94, 2000. doi: 10.1145/349299.349314. URL <https://doi.org/10.1145/349299.349314>.
- Thomas Parr, Giovanni Pezzulo, and Karl J. Friston. *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. MIT Press, 2022. ISBN 9780262045368. URL <https://mitpress.mit.edu/9780262045368/active-inference/>.
- Belinda Phipson and Gordon K. Smyth. Permutation P-values should never be zero: calculating exact P-values when permutations are randomly drawn. *Statistical Applications in Genetics and Molecular Biology*, 9(1):Article 39, 2010. doi: 10.2202/1544-6115.1585. URL <https://gksmyth.github.io/pubs/PermPValuesPreprint.pdf>.
- Qiling Framework Project. Qiling framework, 2026. URL <https://github.com/qilingframework/qiling>.
- Nguyen Anh Quynh. Capstone: Next-gen disassembly framework. Black Hat USA 2014 conference slides, 2014. URL <https://www.capstone-engine.org/BHUSA2014-capstone.pdf>.
- Nguyen Anh Quynh and Dang Hoang Vu. Unicorn: Next generation cpu emulator framework. Black Hat USA 2015 conference slides, 2015. URL <https://www.unicorn-engine.org/BHUSA2015-unicorn.pdf>.
- Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. doi: 10.1109/5.18626. URL <https://ieeexplore.ieee.org/document/18626>.
- radare2 Project. radare2, 2026. URL <https://rada.re/>.
- ReactiveBayes. Using rxinfer from python, 2026a. URL <https://docs.rxinfer.com/stable/manuals/how-to-use-rxinfer-from-python/>.
- ReactiveBayes. Static inference, 2026b. URL <https://docs.rxinfer.com/stable/manuals/inference/static/>.
- Rizin Project. Rizin reverse engineering framework, 2026. URL <https://rizin.re/>.
- RxInfer contributors. Bethe free energy implementation in RxInfer. RxInfer v5.4.0 documentation, 2026a. URL <https://github.com/ReactiveBayes/RxInfer.jl/blob/b4335792b4c8fbb2cdb5b560491f2443c97e7e35/docs/src/manuals/variational/bethe-free-energy.md>. Source revision b4335792b4c8fbb2cdb5b560491f2443c97e7e35.
- RxInfer contributors. Message passing. RxInfer v5.4.0 documentation, 2026b. URL <https://github.com/ReactiveBayes/RxInfer.jl/blob/b4335792b4c8fbb2cdb5b560491f2443c97e7e35/docs/src/concepts/message-passing.md>. Source revision b4335792b4c8fbb2cdb5b560491f2443c97e7e35.
- Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLoS Computational Biology*, 9(10):e1003285, 2013. doi: 10.1371/journal.pcbi.1003285. URL <https://doi.org/10.1371/journal.pcbi.1003285>.

- Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011. ISBN 9781107003637. doi: 10.1017/CBO9780511777110. URL <https://www.cambridge.org/core/books/introduction-to-bisimulation-and-coinduction/8B54001CB763BAE9C4BA602C0A341D60>.
- Eric Schulte, Jonathan Dorn, Antonio Flores-Montoya, Aaron Ballman, and Tom Johnson. Gtirb: Intermediate representation for binaries, 2019. URL <https://arxiv.org/abs/1907.02859>.
- Peter Sefton, Eoghan Ó Carragáin, Stian Soiland-Reyes, Oscar Corcho, Daniel Garijo, Raul Palma, Frederik Coppens, Carole Goble, et al. Ro-crate metadata specification 1.1.3. Research Object Crate Recommendation, 2023. URL <https://www.researchobject.org/ro-crate/specification/1.1/>.
- Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Sok: (state of) the art of war: Offensive techniques in binary analysis. In *2016 IEEE Symposium on Security and Privacy*, pages 138–157, 2016. doi: 10.1109/SP.2016.17. URL https://sites.cs.ucsb.edu/~vigna/publications/2016_SP_angrSoK.pdf.
- The Rust Project Developers. The rustc book: Command-line arguments and code generation, 2026. URL <https://doc.rust-lang.org/rustc/command-line-arguments.html>.
- Trail of Bits. Mcsema executable lifter, 2026. URL <https://github.com/lifting-bits/mcsema>.
- Unicorn Engine Project. Unicorn: The ultimate cpu emulator, 2026. URL <https://www.unicorn-engine.org/>.
- Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Paul Groth, Carole Goble, Jeffrey S. Grethe, et al. The fair guiding principles for scientific data management and stewardship. *Scientific Data*, 3:160018, 2016. doi: 10.1038/sdata.2016.18. URL <https://doi.org/10.1038/sdata.2016.18>.