Tacit Verification Apprenticeship for Self-Evolving Scientific Agents

David Scott Lewis [1] **Karl Wang** [1] **Zhaoxiang Feng** [1] **Junhan Wang** [1] **Saien Deng** [1]

Abstract

Self-evolving scientific agents will not be reliable merely because they can call proof assistants, validators, model checkers, or executable workflows. The hard part is a missing layer of tacit verification craft: choosing definitions, preserving informal intent, splitting proof obligations, navigating libraries, repairing failed scripts, and knowing when a formally accepted artifact is scientifically misaligned. We introduce *Tacit Verification Apprenticeship* (TVA), an architecture and evaluation agenda in which agents learn this craft from proof-state trajectories, failed attempts, expert corrections, semantic-alignment checks, and domain validation traces. We instantiate the idea with a typed memory schema, utility model, failure taxonomy, benchmark design, and three reproducible corpus probes over 1,406 curated records and an executable semantic-alignment demo, showing that craft dimensions are recoverable, apprenticeship retrieval improves with process annotations, repair-memory transfer requires explicit gates, and the sentinel can convert semantic disagreement into repair or abstention. The result is a practical blueprint for scientific agents that improve across cycles while remaining checkable, auditable, and semantically faithful.

1. Introduction

Large language models have made formal theorem proving, autoformalization, and scientific-agent workflows more programmable, but the resulting systems remain brittle at the boundary between informal intent and checkable artifact. A model can emit a Lean statement, a Coq script, a model-checking specification, or a Python validation workflow; it can also pass a syntactic checker while omitting the condition that made the original claim meaningful. The central risk is therefore not only hallucination. It is the production of artifacts that are formally or operationally acceptable under a narrow oracle but misaligned with the mathematical or scientific task.

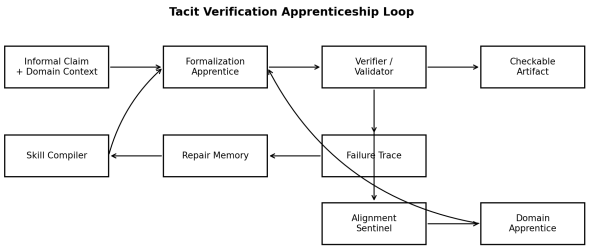

Figure 1. The TVA loop. Verification failure is treated as training data: traces are localized, stored in repair memory, distilled into skills, checked by an alignment sentinel, and reused under explicit transfer boundaries.

This paper argues that self-evolving scientific agents need a learnable layer of *tacit verification craft*. Expert formalizers do not merely know syntax. They decide which definitions make induction possible, when to introduce an auxiliary lemma, how to read a proof state, which library theorem is worth importing, whether a tactic script is too brittle, how to interpret a counterexample, and when a proof has verified the wrong statement. Existing work provides important pieces: Lean and Lean 4 as formal substrates (de Moura et al., 2015a; de Moura & Ullrich, 2021); retrieval-augmented proving and proof environments (Yang et al., 2023); neural and LLM-based autoformalization (Wang et al., 2018; Wu et al., 2022; Azerbayev et al., 2023); semantic-alignment checks (Lu et al., 2024; Zhou et al., 2024a); proof engineering and repair (Klein, 2014; First et al., 2023); scientific V&V (Oberkampf & Roy, 2010); and cognitive apprenticeship (Collins et al., 1988; Brown et al., 1989; Collins et al., 1991). What is missing is an agent architecture that treats the invisible craft connecting these literatures as the object of learning.

Contributions. We make four contributions. First, we define the *Tacit Verification Bottleneck*: the gap between explicit formal artifacts and the craft needed to create, diagnose, repair, and transfer them. Second, we propose TVA, a modular architecture with five components: a Formalization

[1]AIXC Research, Zaragoza, Spain. Correspondence to: David Scott Lewis <reports@aixcresearch.com>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

Apprenticeship Agent, Proof/Validation Repair Memory, Tacit Skill Compiler, Semantic Alignment Sentinel, and Domain Verification Apprentice. Third, we give a typed representation schema and algorithms for skill utility, failure diagnosis, repair selection, apprenticeship update, and transfer gating. Fourth, we introduce ApprenticeshipBench-SESA, a benchmark design spanning formal math, semantic alignment, proof repair, executable scientific specs, and domain validation, and report three corpus probes plus an executable semantic-sentinel demo that stress-test the design assumptions.

2. The Tacit Verification Bottleneck

Definition 2.1 (Tacit Verification Bottleneck)**.** The bottleneck is the systematic loss of expert craft when informal mathematical or scientific intent is converted into checkable artifacts. The lost information includes design rationales, rejected definitions, failed tactics, unstated side conditions, library-search paths, alignment doubts, counterexamples, and domain-risk judgments.

Formal systems expose strong feedback, but their feedback is incomplete. A proof assistant can reject an ill-typed tactic but cannot by itself certify that the theorem statement captured the informal claim. A model checker can produce a counterexample but cannot decide whether the intended model class was too weak. A scientific workflow can pass tests while leaking labels or violating domain assumptions. Program repair teaches a parallel lesson: generate-and-validate systems can overfit incomplete tests (Smith et al., 2015), and patch quality depends on fault localization, context, and independent oracles (Goues et al., 2019; Xia et al., 2023; Liu et al., 2019; Koyuncu et al., 2020).

TVA treats these craft types as first-class skills. Each skill s_i is stored with preconditions, postconditions, failure signatures, repair operators, provenance, and transfer boundaries. Given memory K_t, goal G_t, execution trace E_t, failure trace F_t, and domain context D_t, we score a candidate skill by

$$U(s_i; G_t, D_t, K_t) = \alpha p_{\text{succ}} + \beta r_{\text{repair}} + \gamma f_{\text{sem}} \tag{1}$$

$$+ \delta v_{\text{reuse}} + \eta \tau_{\text{transfer}} - \lambda c_{\text{verify}} \tag{2}$$

$$- \mu b_{\text{brittle}} - \rho \,\text{risk}(D_t). \tag{3}$$

The transfer term is deliberately gated: a local heuristic generalizes only when the current goal, domain assumptions, oracle strength, and failure signature are all within the stored boundary of prior successful use.

Utility instantiation. In the implementation used by our probes, every component of U is normalized to $[0, 1]$. p_{succ} is the smoothed past validation rate for the skill under a matching oracle; r_{repair} is the observed reduction in recurrent failures after applying the repair; f_{sem} is the alignment-sentinel score; v_{reuse} measures held-out reuse; τ_{transfer} is zero unless craft type, domain family, oracle strength, and excluded cases satisfy the stored transfer boundary; c_{verify}, b_{brittle}, and $\text{risk}(D_t)$ penalize expensive checks, perturbation failures, and domain stakes. A skill is eligible only if the hard transfer gate passes and $U > 0$; otherwise the selector abstains or asks for a stronger oracle. Appendix F gives the concrete scoring card.

Table 1. Tacit craft types operationalized as agent-learning targets.

Craft	Observable trace	Feedback signal
Statement design	edits to quantifiers, hypotheses, goals	counterexample or proof gap
Definition choice	representation swaps, coercions, instances	proof length, brittleness
Lemma decomposition	subgoal tree, auxiliary claims	reuse and closure rate
Library navigation	imports, retrieved premises, failed searches	premise hit rate
Tactic selection	proof-state deltas per tactic	progress and failure class
Proof repair	error, patch, recheck loop	verified repair success
Semantic alignment	NL-formal pairs, paraphrases, tests	faithfulness score
Executable spec	tests, traces, model-checking witnesses	validation coverage
Domain formalization	causal, finance, bio, physics constraints	domain invariant checks
Failure distillation	normalized failure signatures	future retrieval utility

Design principles. The bottleneck has four practical consequences. *First*, agents should log negative evidence. A failed tactic, rejected formalization, or counterexample-guided repair is not clutter; it is the primary evidence from which a future agent learns what not to try. *Second*, agents should separate syntactic success from semantic success. A theorem can typecheck while omitting an intended hypothesis, and an executable workflow can pass tests while validating the wrong scientific model. *Third*, transfer should be earned. A proof-repair pattern learned in algebra need not transfer to causal identification, and a no-arbitrage repair need not transfer to biological pathway validation. *Fourth*, agents should maintain auditable skill provenance: every compiled skill should point back to the traces, validators, and critiques that licensed it.

From craft to supervision. Each craft type creates a different supervision signal. Statement design is supervised by counterexamples, failed proof obligations, and semantic-diff checks. Definition choice is supervised by proof length, elaboration failures, blocked induction, and brittleness under refactoring. Lemma decomposition is supervised by subgoal

closure and later reuse. Library navigation is supervised by premise-hit rate and import minimization. Tactic selection is supervised by proof-state deltas and proof-progress predictors. Proof repair is supervised by revalidation after patches. Semantic alignment is supervised by paraphrase checks, behavioral examples, and formal equivalence or implication tests. Executable specification is supervised by tests, model-checking witnesses, runtime monitors, and independent validation data. Domain formalization is supervised by domain invariants such as causal assumptions, no-arbitrage constraints, ontology consistency, dimensional analysis, and numerical enclosure. Failure distillation is supervised by later retrieval utility: a failure label is good only if it helps a future agent choose a better repair.

Why apprenticeship rather than prompting. Prompting can elicit isolated behaviors but does not by itself create durable craft. Apprenticeship requires a curriculum of situated attempts, visible expert reasoning, correction, reflection, and fading of support. In human proof education, making goal structure visible changes how novices write proofs; in proof assistants, the goal state and tactic trace make that visibility executable. TVA therefore treats every formalization episode as a training episode with observable moves, feedback, and responsibility transfer.

3. Architecture

TVA is an apprenticeship architecture rather than a single prover. The *Formalization Apprenticeship Agent* observes expert demonstrations, generated attempts, proof states, corrections, and verifier feedback. It decomposes informal intent into candidate statements and proof obligations, queries libraries, emits proof sketches, and records each failed or successful step. The *Proof/Validation Repair Memory* stores normalized failures such as missing hypotheses, brittle tactics, import breaks, causal graph mismatches, no-arbitrage omissions, ontology mismatches, and numerical enclosure gaps. The *Tacit Skill Compiler* abstracts repeated traces into reusable skills: "introduce a monotonicity lemma before invoking simplification," "repair a causal claim by adding an unobserved-confounder assumption," or "replace a fragile tactic chain with a structural proof." The *Semantic Alignment Sentinel* screens for provable-but-wrong artifacts by comparing informal intent, formal statement, tests, paraphrases, witnesses, and domain invariants. The *Domain Verification Apprentice* specializes these procedures for systems science, causality, finance, bioinfo., physics, and adjacent domains.

The schema in Table 2 is the minimum substrate. It unifies proof-assistant traces, program-repair patch records, semantic-alignment checks, and scientific validation logs. It is intentionally typed enough to support retrieval and transfer gating, but lightweight enough to be logged by current agent systems.

Algorithm 1 Tacit Verification Apprenticeship Update

Require: Goal G_t, domain D_t, memory K_t, verifier V
Retrieve candidate skills $S \leftarrow \mathrm{Retrieve}(K_t, G_t, D_t)$
Rank S by $U(s_i; G_t, D_t, K_t)$ with transfer gates
Produce formal artifact A and rationale R using top-ranked skills
Run verifier/validator: $(y, E_t, F_t) \leftarrow V(A)$
Run alignment sentinel: $z \leftarrow \mathrm{Align}(A, R, G_t, D_t)$
if y succeeds and z is aligned **then**
  Store success trace, provenance, and reusable proof obligations
else
  Localize failure, propose repair, and store normalized signature
  Compile or update skills whose repair value is positive
end if
Ensure: Updated tacit skill memory K_t^+

Alignment-sentinel instantiation. The sentinel is a committee of independent checks rather than a single classifier. It extracts semantic facets from the informal goal (objects, quantifier order, assumptions, conclusion, examples, and domain invariants), compares them with the formal artifact, generates discriminating examples or counterexamples when possible, and asks whether the artifact is equivalent to, stronger than, weaker than, or incomparable with the intent. We instantiate its score as $f_{\mathrm{sem}} = \min(q_{\mathrm{facet}}, q_{\mathrm{example}}, q_{\mathrm{back}}, q_{\mathrm{domain}})$ so that a high score requires agreement across facet matching, executable probes, backtranslation/paraphrase, and domain checks. Low confidence produces a repair request or abstention, not silent acceptance.

Interface contracts. The five modules communicate through typed objects rather than free-form chat. The apprentice accepts a goal object and returns a candidate artifact plus a rationale. The verifier returns a typed execution trace, not merely a Boolean. Repair memory indexes failures by craft cause, domain, local context, witness, and repair delta. The skill compiler converts repeated repairs into parametric operators only after they have passed an independent alignment or validation check. The sentinel can veto a formally accepted artifact when the informal statement, examples, or domain invariants disagree with the formal artifact. These contracts are what make the architecture self-evolving without making it self-justifying.

Memory policy. Memory is divided into episodic traces, compiled skills, and governance records. Episodic traces

Table 2. Core representation objects. Full JSON-style schemas are in Section C.

Object	Essential fields
Goal	informal statement, domain, target artifact, risk level
Formal statement	system, imports, declarations, theorem text
Proof state	goals, hypotheses, local context, library context
Tactic trace	action, pre-state, post-state, dependencies
Failure signature	diagnostic, localized object, witness, oracle
Repair action	edit script, rationale, validation result
Intended semantics	paraphrase, tests, invariants, examples
Transfer boundary	preconditions, excluded domains, known brittleness
Confidence	typecheck, semantic, domain, calibration evidence

preserve raw proof states, failed commands, counterexamples, and human critiques. Compiled skills summarize recurring repair operators with preconditions and postconditions. Governance records store abstentions, non-transfer cases, oracle weaknesses, and version incompatibilities. The third category is essential: without explicit records of where a skill failed to transfer, self-evolution becomes a mechanism for propagating confident mistakes.

Domain specialization. Domain verification apprentices do not replace proof assistants; they add domain-specific validation oracles. A causal apprentice checks whether a graph and identification claim preserve assumptions such as ignorability and conditional independence. A finance apprentice checks no-arbitrage, conservation, rounding, and oracle-manipulation invariants. A bioinfo. apprentice checks identifier normalization, ontology alignment, leakage, and statistical validity. A physics apprentice checks dimensional consistency, approximations, numerical enclosures, and provenance from diagrams or simulations. These oracles become part of the same repair loop as theorem-prover feedback.

4. Experiments

We ran four deterministic experiments over the curated project corpus and a small executable semantic-alignment demo. These are not claims about end-to-end theorem-proving performance; they are design probes for whether the proposed memory and apprenticeship structures are measurable with available public-paper metadata and extracted annotations. The corpus contains 1,406 raw JSON records and 987 deduplicated title-normalized records; the seven top-cluster files contribute 450 deduplicated records after removing metadata artifacts. The corpus probes use TF-IDF, weak labels from corpus categories, and a structured failure matrix; the demo uses finite executable predicates to test and repair semantic alignment end to end.

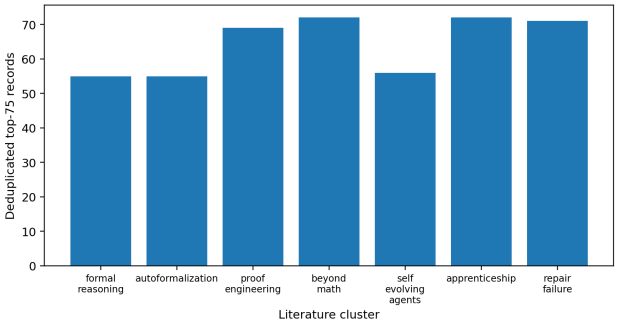

Figure 2. Deduplicated top-cluster corpus used for the experiments. The corpus is balanced enough to probe cross-literature craft signatures, but it is not a benchmark for theorem-prover success.

Table 3. Summary of the four reproducible experiments.

Experiment	Comparison	Primary result	Artifact
Craft probe	7-way corpus	acc. 0.93, macro-F1 0.93	Fig. 3
Retrieval	annotations vs title	MRR 1.00 vs 0.94	Fig. 4
Repair memory	gated vs signature	acc. 0.62 vs 0.12	Fig. 5
Sentinel demo	finite NL-formal tasks	F1 1.00; repairs 9/12	App. G

The corpus is deliberately heterogeneous. It contains formal-reasoning systems, autoformalization, proof-engineering practice, scientific formalization beyond pure mathematics, self-evolving agents, cognitive apprenticeship, and repair/failure learning. That heterogeneity is a feature: TVA is designed for agents that must move between proof assistants, validators, domain workflows, and human feedback. It is also a limitation: the labels are weak and inherited from file categories, and the experiments should be interpreted as architecture probes rather than definitive empirical claims.

Experiment 1: craft recoverability. We defined twelve craft descriptions, embedded papers and craft descriptions in the same TF-IDF space, and measured cluster-wise craft similarity. A 5-fold linear classifier predicts the seven top-level literature clusters from title, abstract, TLDR, and importance text with mean accuracy 0.93 ± 0.02 and macro-F1 0.93 ± 0.02. Kruskal-Wallis tests over craft scores show strong between-cluster differences for every craft dimension (median $p = 4.9 \times 10^{-18}$). The heatmap in Figure 3 is not a semantic oracle, but it supports the design premise that craft signals are visible in process-rich literature.

Experiment 2: apprenticeship retrieval. We queried the top-cluster corpus with the twelve craft definitions and

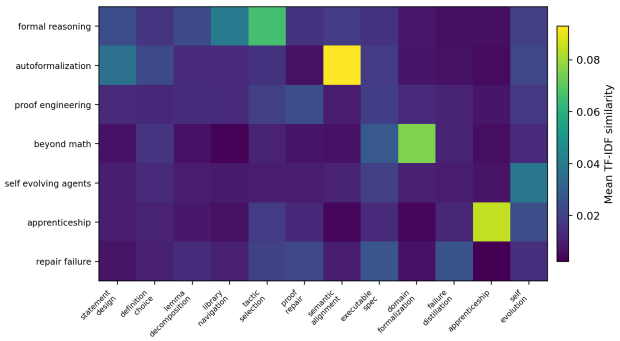

Figure 3. Experiment 1: mean craft-similarity profiles for the deduplicated top-cluster corpus. Distinct craft dimensions align with different literatures, while definition choice and domain transfer remain comparatively diffuse.

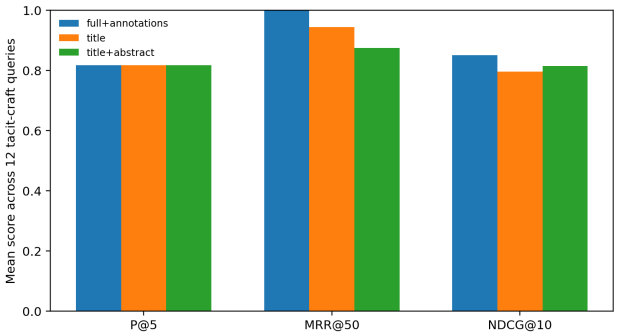

Figure 4. Experiment 2: retrieval ablation for tacit-craft queries. Process annotations improve rank quality over title-only and title-plus-abstract indexes.

evaluated retrieval against weak cluster relevance. Title-only retrieval is already strong because titles are topical, but full process annotations improve mean NDCG@10 from 0.80 to 0.85 and MRR@50 from 0.94 to 1.00. This supports a practical prescription: proof-state traces, failed attempts, rationales, and "why important" fields should be logged because they are the material from which an apprentice retrieves prior craft.

Experiment 3: repair-memory transfer. We built a 26-case failure matrix covering math, causality, finance, bioinfo., physics, and general agent failures. Leave-one-failure-out nearest-neighbor retrieval predicts a coarse repair operator. Signature-only memory obtains 0.12 accuracy and has a 0.58 unsafe-transfer rate. Adding craft and domain context raises accuracy to 0.73 and eliminates unsafe transfers; a stricter transfer-gated variant obtains 0.62 accuracy with zero unsafe transfers. The result is expected but important: memory is useful only when failures are stored with craft and domain boundaries, not just surface diagnostics.

Experiment 4: executable sentinel demo. To ground the alignment module, we added an end-to-end finite-domain demo with 18 informal arithmetic claims, candidate executable formalizations, and hidden reference predicates. A syntax-only baseline accepts all 18 candidates and therefore misses 12 semantic errors. The sentinel compares truth sets over finite domains, localizes mismatch type, and applies repair templates when available; it detects all 12 misalignments with no false positives, repairs 9, and abstains on 3 cases outside the repair library. This is a toy oracle, not a proof-assistant result, but it exercises the intended loop: detect provable-but-wrong artifacts, repair when licensed, and abstain when the transfer boundary is insufficient.

Table 4. Representative failure modes that TVA stores as repair-memory cases.

Failure	Signature	Repair memory update
Misformalized quantifier	formal theorem weakens intent	store statement-alignment repair
Missing hypothesis	proof requires unstated side condition	add side-condition template
Bad definition	induction or reuse blocked	store representation refactor
Missing lemma	repeated subgoal pattern	compile lemma-decomposition skill
Library-blind search	known theorem not retrieved	update premise-retrieval index
Brittle tactic	breaks under minor version change	prefer structural proof pattern
Causal mismatch	confounder absent from DAG	add domain-assumption repair
No-arbitrage omission	invariant permits free lunch	add finance invariant check
Ontology mismatch	entity identifiers disagree	add ontology-alignment step
Numerical gap	missing enclosure bound	require certified bound
Oracle overfit	patch passes shallow tests	add discriminating test
Memory overtransfer	repair used in wrong domain	tighten transfer boundary

What the experiments imply. Experiment 1 supports the hypothesis that craft categories are recoverable from literature traces, but also shows that some craft types are diffuse. This matters because diffuse craft is precisely what tends to be omitted from benchmark datasets. Experiment 2 shows that process annotations improve retrieval, so trace logging should preserve rationales and failed attempts rather than only final proofs. Experiment 3 shows that repair memory must include craft and domain metadata; surface similarity alone retrieves tempting but unsafe analogies. Experiment 4 shows how the sentinel can convert alignment disagreement into a localized repair or abstention. Together, these results support a design in which memory entries are structured, validators are heterogeneous, semantic acceptance is independent of syntactic acceptance, and transfer is explicitly

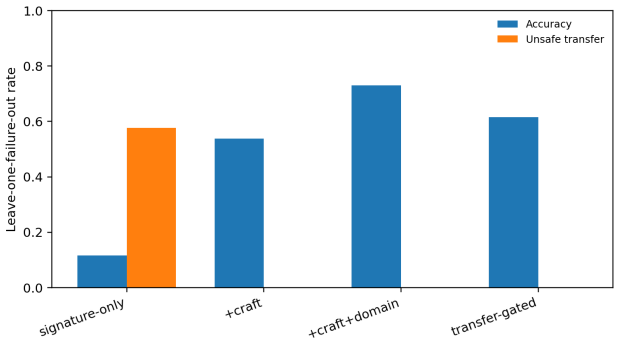

Figure 5. Experiment 3: repair-memory ablation. Surface signatures transfer unsafely; adding craft and domain metadata improves repair selection and suppresses incompatible transfer.

governed.

5. Benchmark and Evaluation

ApprenticeshipBench-SESA evaluates whether an agent improves at verification craft, not merely whether it solves isolated problems. It contains six task families: semantic alignment, proof repair, lemma decomposition, executable scientific specifications, domain validation, and cross-domain transfer. In formal mathematics, tasks should draw from Lean, Isabelle, Rocq/Coq, MiniF2F-style theorem proving, ProofNet-style statement alignment, and library-maintenance scenarios. In adjacent domains, tasks should include causal-graph validation, no-arbitrage and price-oracle invariants, bioinfo. ontology and leakage checks, verified scientific computing, and multimodal physics reasoning.

The main metrics are *Repair Success*, *Semantic Faithfulness*, *Lemma Reuse*, *Brittleness Reduction*, *Human Correction Efficiency*, *Cross-Domain Transfer*, and *Skill Improvement Slope*. A result counts only if the oracle matches the task: proof assistants for formal proofs, model checkers or SMT solvers for executable specs, independent tests for repair, and domain-specific validators for scientific claims. Because repair systems can overfit incomplete oracles (Smith et al., 2015), benchmark splits must include perturbations, independent witnesses, version changes, and semantically equivalent but syntactically different goals.

Task design. Each benchmark item should include an initial informal claim, a target artifact type, a toolchain version, at least one known failure trace, and hidden or independent checks. A proof-repair item might provide a broken Lean or Rocq proof after a library update. A semantic-alignment item might provide an informal theorem and two formal statements, only one of which preserves the intended quantifier structure. A causal item might provide a natural-language study design and ask the agent to encode the graph, identify estimands, and flag untestable assumptions. A finance item might provide an executable market mechanism and ask for a no-arbitrage or conservation invariant. A bioinfo. item might ask for a validated workflow that handles ontology mapping and split hygiene. A physics item might require dimension checks and verified numerical bounds.

Table 5. Main evaluation axes for ApprenticeshipBench-SESA.

Metric	Question answered
Repair Success	Does the agent repair failures under the right oracle?
Semantic Faithfulness	Did the artifact preserve the intended claim?
Lemma Reuse	Did compiled skills reduce future proof burden?
Brittleness Reduction	Does the solution survive perturbations and versions?
Human Correction Efficiency	Did apprenticeship reduce expert edits?
Cross-Domain Transfer	Did the skill transfer within its boundary?
Skill Improvement Slope	Does performance improve across episodes?

Table 6. Example ApprenticeshipBench-SESA task instances and primary oracle.

Task	Agent action	Primary oracle
Lean TCS proof repair	repair broken lemma after dependency change	Lean checker + hidden theorem
Semantic alignment	choose faithful formalization of NL theorem	equiv. / impl. tests
Lemma decomposition	propose reusable auxiliary claims	later proof reuse
Causal validation	encode graph and flag missing assumptions	CI / ID checks
Finance invariant	add no-arbitrage or conservation spec	model checker/SMT
Bio workflow	repair ontology and leakage failures	held-out validation
Physics derivation	check units and numerical enclosure	dimension and interval oracle

Ablations. The benchmark should compare no-memory agents, episodic-memory agents, compiled-skill agents, alignment-sentinel agents, and transfer-gated agents. The key claim is not that every module always improves raw pass rate. It is that a self-evolving agent should become more reliable across repeated episodes: fewer repeated mistakes, fewer semantically wrong proofs, fewer unsafe transfers, and fewer human edits.

6. Related Work and Positioning

Theorem-proving systems and proof assistants provide the executable substrate for TVA (de Moura et al., 2015a; de Moura & Ullrich, 2021; Qian et al., 2025). Retrieval-augmented provers and proof-search systems make premise selection, tactic generation, and proof-state interaction learn-

able (Wang et al., 2017; Yang et al., 2023; George et al., 2025; Samoylov & Vosoughi, 2025). Autoformalization work addresses the informal-to-formal interface (Wang et al., 2018; Wu et al., 2022; Azerbayev et al., 2023; Jiang et al., 2023a), while semantic-alignment work targets the danger of formally accepted but mis-specified statements (Lu et al., 2024; Zhou et al., 2024a). Program repair contributes fault localization, patch generation, validation, and overfitting lessons (Goues et al., 2019; Wen et al., 2018; Xia et al., 2023; Zhang et al., 2023). Scientific-agent work contributes self-evolution, reflection, memory, and long-horizon workflow execution (Gottweis et al., 2025; Liang et al., 2025; Zhou et al., 2024b; Chen et al., 2024; Fang et al., 2025). Cognitive apprenticeship explains why tacit craft must be made visible through modeling, coaching, scaffolding, fading, articulation, and reflection (Collins et al., 1988; Brown et al., 1989; Collins et al., 1991; Dennen & Burner, 2007). TVA connects these threads into a single learning target: durable verification craft.

Novelty relative to prior systems. The closest theorem-proving systems learn tactics, retrieve premises, or synthesize proofs; the closest autoformalization systems translate and align statements; the closest repair systems generate patches; the closest scientific agents maintain memory and run workflows. TVA is different in its unit of learning. The learned object is not only a proof, patch, prompt, or workflow, but a typed verification skill with a failure signature, repair operator, semantic-faithfulness evidence, and transfer boundary. That distinction is important for self-evolution: if the memory records only successes, the agent cannot learn where the success should not generalize.

Scientific risk. In empirical domains, formalization can create a false sense of security. A causal graph can be precisely encoded while omitting the relevant confounder; a finance invariant can be checked under an unrealistic market model; a bioinfo. workflow can be reproducible while leaking labels; a physics derivation can typecheck while using the wrong approximation regime. TVA addresses this by requiring the domain apprentice and alignment sentinel to preserve the distinction between mathematical verification, executable validation, and scientific adequacy.

7. Operational Consequences

A practical TVA implementation should log more than prompts and final answers. For each episode it should store the informal statement, formal artifact, proof-state sequence, available imports, retrieved premises, rejected candidates, verifier diagnostics, counterexamples, semantic-alignment checks, human critique, final repair, and the reason for accepting or abstaining. Failed repairs are especially valuable: they define the boundary of a skill and prevent memory from becoming a catalogue of overconfident successes.

The training schedule should follow cognitive apprenticeship. Early episodes expose full expert traces; middle episodes hide some scaffolds and ask the agent to diagnose failures or choose repair operators; later episodes provide only the informal goal and validators. Support should be reintroduced when a repeated failure appears. In safety-critical or high-risk scientific domains, calibrated abstention is a successful outcome when semantic faithfulness, oracle strength, or transfer validity is weak.

8. Limitations and Conclusion

The experiments are corpus, memory, and finite-oracle probes, not live proof-assistant training runs. Their labels are weak, their retrieval model is deliberately simple, and the repair-memory task is a structured simulation. The executable sentinel demo verifies the control flow of semantic detection and repair, but it does not replace Lean/Rocq/Isabelle evaluation. Next empirical work should instantiate the schema on proof-assistant logs such as LeanDojo and Baldur-style traces, mathlib version perturbations, model-checking counterexamples, and scientific validation notebooks; connect the sentinel to ATP/SMT oracles; and measure reliability gains in live systems rather than only corpus probes.

Testable ablations. The thesis makes concrete empirical predictions. It is weakened if a one-shot prover with no trace memory, no alignment check, and no transfer boundary matches a memory-augmented apprentice on pass rate, perturbation robustness, semantic faithfulness, and human-correction efficiency. Required ablations should remove repair memory, craft labels, domain risk, independent semantic tests, and transfer gates, then measure not only immediate proof success but the slope of improvement over episodes.

A stronger base prover remains valuable, but it does not eliminate the apprenticeship problem. Scaling can increase the probability of finding a proof term, tactic sequence, or validation script; it does not necessarily preserve the relation between generated artifact and informal claim, explain which failed attempts should become reusable skills, or decide which domain assumptions forbid transfer. TVA therefore complements stronger generation models: the verifier supplies hard feedback, the generator proposes candidates, and the apprenticeship layer records why a candidate worked, why it failed, and where it should not be reused.

The main claim is that self-evolution without verification apprenticeship is unstable. Agents that learn only to generate artifacts will eventually learn to satisfy shallow oracles. Agents that learn tacit verification craft can instead accu-

Table 7. Minimal reporting checklist for a verification-apprenticeship episode.

Record	Why it is needed
Informal intent	preserves the claim against which formal artifacts are judged
Candidate artifact	records the theorem, specification, proof script, or workflow
Verifier trace	exposes proof states, diagnostics, witnesses, and oracle strength
Rejected attempts	prevents repeated failures and teaches non-transfer cases
Semantic checks	distinguishes typechecked artifacts from faithful artifacts
Repair delta	connects localized failure to the edit that fixed or failed to fix it
Human critique	converts expert judgment into reusable apprenticeship signal
Transfer boundary	states where a repair may not be reused safely

mulate reusable, auditable knowledge about how to make claims checkable, how to repair them when they fail, and how to know when a locally successful repair should not transfer. TVA supplies the architecture, representations, algorithms, and benchmark agenda for that transition.

Reporting standard. A reported success should be traceable to a proof object, validation script, counterexample check, or expert correction. A compiled skill should include a compact card: what it does, where it was learned, what failed attempts shaped it, what oracles validated it, and which domains or toolchain versions are excluded. These cards make the system inspectable and give maintainers a mechanism to delete, quarantine, or downgrade a learned skill after a library change, benchmark leak, or domain error is discovered.

Curriculum implication. The useful training examples are not only solved proofs. A curriculum should deliberately include near misses: theorem statements differing by one quantifier, definitions that are extensionally equivalent but proof-hostile, library searches where the obvious query fails, and repairs that pass a shallow check while failing an independent semantic check. These cases teach the agent that verification craft is the disciplined management of evidence, not greedy satisfaction of the nearest oracle.

References

Azerbayev, Z., Piotrowski, B., Schoelkopf, H., Ayers, E. W., Radev, D., and Avigad, J. ProofNet: Autoformalizing and Formally Proving Undergraduate-Level Mathematics, 2023. URL https://doi.org/10.48550/arxiv.2302.12433.

Beckert, B., Grebing, S., and Bohl, F. A Usability Evaluation of Interactive Theorem Provers Using Focus Groups, 2015. URL https://doi.org/10.1007/978-3-319-15201-1_1.

Bobbin, M. P., Sharlin, S., Feyzishendi, P., Dang, A. H., Wraback, C. M., and Josephson, T. R. Formalizing chemical physics using the Lean theorem prover, 2023. URL https://doi.org/10.1039/d3dd00077j.

Brown, J. S., Collins, A., and Duguid, P. Situated Cognition and the Culture of Learning, 1989. URL https://doi.org/10.3102/0013189x018001032.

Chao, L. P., Tumer, I. Y., and Ishii, K. Design Process Error-Proofing: Engineering Peer Review Lessons From NASA, 2004. URL https://doi.org/10.1115/detc2004-57764.

Chen, Z., Chen, S., Ning, Y., Zhang, Q., Wang, B., Yu, B., Li, Y., Liao, Z., Wei, C., Lu, Z., Dey, V., Xue, M., et al. ScienceAgentBench: Toward Rigorous Assessment of Language Agents for Data-Driven Scientific Discovery, 2024. URL https://doi.org/10.48550/arxiv.2410.05080.

Collins, A. Cognitive Apprenticeship and Instructional Technology, 1988. URL https://doi.org/10.21236/ada203609.

Collins, A. Cognitive apprenticeship : Teaching the craft of reading, writing, and mathematics, 1989. URL http://oai.dtic.mil/oai/oai?&verb=getRecord&metadataPrefix=html&identifier=ADA178530.

Collins, A., Brown, J. S., and Newman, S. E. Cognitive Apprenticeship, 1988. URL https://doi.org/10.5840/thinking19888129.

Collins, A., Brown, J. S., and Holum, A. COGNITIVE APPRENTICESHIP: MAKING THINKING VISIBLE, 1991. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.8616.

Collins, A., Brown, J. S., and Newman, S. E. Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing, and Mathematics, 2018. URL https://doi.org/10.4324/9781315044408-14.

Cunningham, G., Bunescu, R., and Juedes, D. Towards Autoformalization of Mathematics and Code Correctness: Experiments with Elementary Proofs, 2022. URL https://doi.org/10.18653/v1/2022.mathnlp-1.4.

Damani, M., Luo, Z., Wenzel, E., and Sartoretti, G. PRIMAL_2: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning - Lifelong, 2021. URL https://doi.org/10.1109/lra.2021.3062803.

de Moura, L. and Ullrich, S. The Lean 4 Theorem Prover and Programming Language, 2021. URL https://doi.org/10.1007/978-3-030-79876-5_37.

de Moura, L., Kong, S., Avigad, J., van Doorn, F., and von Raumer, J. The Lean Theorem Prover (System Description), 2015a. URL https://doi.org/10.1007/978-3-319-21401-6_26.

de Moura, L., Kong, S., Avigad, J., van Doorn, F., and von Raumer, J. The Lean Theorem Prover, 2015b. URL https://www.andrew.cmu.edu/user/avigad/Talks/clarke.pdf.

Dennen, V. P. and Burner, K. J. The Cognitive Apprenticeship Model in Educational Practice, 2007. URL https://doi.org/10.4324/9780203880869.ch34.

Dietrich, D., Whiteside, I., and Aspinall, D. Polar: A Framework for Proof Refactoring, 2013. URL https://doi.org/10.1007/978-3-642-45221-5_52.

Fang, J., Peng, Y., Zhang, X., Wang, Y., Yi, X., Zhang, G., Xu, Y., Wu, B., Liu, S., Li, Z., Ren, Z., Aletras, N., et al. A Comprehensive Survey of Self-Evolving AI Agents: A New Paradigm Bridging Foundation Models and Lifelong Agentic Systems, 2025. URL https://doi.org/10.48550/arxiv.2508.07407.

Felty, A. P. and Howe, D. J. Hybrid interactive theorem proving using nuprl and HOL, 1997. URL https://doi.org/10.1007/3-540-63104-6_34.

First, E., Rabe, M. N., Ringer, T., and Brun, Y. Baldur: Whole-Proof Generation and Repair with Large Language Models, 2023. URL https://doi.org/10.1145/3611643.3616243.

George, R. J., Huang, S., Song, P., and Anandkumar, A. LeanProgress: Guiding Search for Neural Theorem Proving via Proof Progress Prediction, 2025. URL https://arxiv.org/abs/2502.17925v3.

Gonthier, G. The Four Colour Theorem: Engineering of a Formal Proof, 2008. URL https://doi.org/10.1007/978-3-540-87827-8_28.

Gottweis, J., Weng, W., Daryin, A., Tu, T., Palepu, A., Sirkovic, P., Myaskovsky, A., Weissenberger, F., Rong, K., Tanno, R., Saab, K., Popovici, D., et al. Towards an AI co-scientist, 2025. URL https://doi.org/10.48550/arxiv.2502.18864.

Goues, C. L., Pradel, M., and Roychoudhury, A. Automated program repair, 2019. URL https://doi.org/10.1145/3318162.

Hammer, R., Ratz, D., Kulisch, U., and Hocks, M. C++ Toolbox for Verified Scientific Computing I: Basic Numerical Problems, 1997. URL http://dl.acm.org/citation.cfm?id=549260.

Hennessy, S. Situated Cognition and Cognitive Apprenticeship: Implications for Classroom Learning, 1993. URL https://doi.org/10.1080/03057269308560019.

Holliday, W. H., Norman, C., and Pacuit, E. Voting Theory in the Lean Theorem Prover, 2021. URL https://doi.org/10.1007/978-3-030-88708-7_9.

Huang, D., Li, H., Li, W., Zhang, H., Dickson, P., Zhan, M., Miller, J. P., Cruchaga, C., Province, M. A., Chen, Y., Payne, P., and Li, F. OmniCellAgent: Towards AI Co-Scientists for Scientific Discovery in Precision Medicine, 2025. URL https://doi.org/10.1101/2025.07.31.667797.

Hurd, J. Formal verification of probabilistic algorithms, 2021. URL https://doi.org/10.48456/tr-566.

Jiang, A. Q., Li, W., and Jamnik, M. Multilingual Mathematical Autoformalization, 2023a. URL https://doi.org/10.48550/arxiv.2311.03755.

Jiang, N., Liu, K., Lutellier, T., and Tan, L. Impact of Code Language Models on Automated Program Repair, 2023b. URL https://doi.org/10.1109/icse48619.2023.00125.

Jojgov, G. G. Incomplete proofs and terms and their use in interactive theorem proving, 2004. URL https://doi.org/10.6100/ir573767.

Kaivola, R. and Kohatsu, K. Proof engineering in the large: formal verification of Pentium4 floating-point divider, 2003. URL https://doi.org/10.1007/s10009-002-0081-6.

Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., and Shimada, S. RoboCup Rescue: search and rescue in large-scale disasters as a domain for autonomous agents research, 2003. URL https://doi.org/10.1109/icsmc.1999.816643.

Klein, G. Proof Engineering Considered Essential, 2014. URL https://doi.org/10.1007/978-3-319-06410-9_2.

Koyuncu, A., Liu, K., Bissyande, T. F. D. A., Kim, D., Klein, J., Monperrus, M., and Le Traon, Y. FixMiner: Mining relevant fix patterns for automated program repair, 2020. URL http://orbilu.uni.lu/handle/10993/44172.

Kumarappan, A., Tiwari, M., Song, P., George, R. J., Xiao, C., and Anandkumar, A. LeanAgent: Lifelong Learning for Formal Theorem Proving, 2024. URL https://arxiv.org/abs/2410.06209v8.

Lawo, C. C-XSC A Programming Environment for Verified Scientific Computing and Numerical Data Processing, 1993. URL https://doi.org/10.1016/s0076-5392(08)62842-8.

Lee, C. D. A Culturally Based Cognitive Apprenticeship: Teaching African American High School Students Skills in Literary Interpretation, 1995. URL https://doi.org/10.2307/748192.

Liang, X., Tao, M., Xia, Y., Wang, J., Li, K., Wang, Y., He, Y., Yang, J., Shi, T., Wang, Y., Zhang, M., and Wang, X. SAGE: Self-evolving Agents with Reflective and Memory-augmented Abilities, 2025. URL https://doi.org/10.1016/j.neucom.2025.130470.

Lincoln, P. and Rushby, J. A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model, 2005. URL https://doi.org/10.1109/ftcsh.1995.532669.

Liu, K., Koyuncu, A., Kim, D., and Bissyande, T. F. TBar: revisiting template-based automated program repair, 2019. URL https://doi.org/10.1145/3293882.3330577.

Lu, J. G., Wan, Y., Huang, Y., Xiong, J., Liu, Z., and Guo, Z. FormalAlign: Automated Alignment Evaluation for Autoformalization, 2024. URL https://doi.org/10.48550/arxiv.2410.10135.

Malecha, G. Extensible proof engineering in intensional type theory, 2015. URL http://nrs.harvard.edu/urn-3:HUL.InstRepos:17467172.

Mitchener, L., Yiu, A., Chang, B., Bourdenx, M., Nadolski, T., Sulovari, A., Landsness, E. C., Barabasi, D. L., Narayanan, S., Evans, N., Reddy, S., Foiani, M., et al. Kosmos: An AI Scientist for Autonomous Discovery, 2025. URL https://doi.org/10.48550/arxiv.2511.02824.

Moons, P., Dou, B., and Desmedt, C. P. Googles AI co-scientist and OpenAIs deep research: new partners in health research?, 2025. URL https://doi.org/10.1093/eurjcn/zvaf066.

Nagatou, K., Nakao, M., and Wakayama, M. VERIFIED NUMERICAL COMPUTATIONS FOR EIGENVALUES OF NON-COMMUTATIVE HARMONIC OSCILLATORS, 2002. URL https://doi.org/10.1081/nfa-120014756.

Oberkampf, W. L. and Roy, C. J. Verification and Validation in Scientific Computing, 2010. URL https://doi.org/10.1017/cbo9780511760396.

Otten, J. leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic (System Descriptions), 2008. URL https://doi.org/10.1007/978-3-540-71070-7_23.

Otten, J. and Bibel, W. leanCoP: lean connection-based theorem proving, 2003. URL https://doi.org/10.1016/s0747-7171(03)00037-3.

Palmer, R. L., Barrus, S., Yang, Y., Gopalakrishnan, G., and Kirby, R. M. Gauss: A Framework for Verifying Scientific Computing Software, 2006. URL https://doi.org/10.1016/j.entcs.2006.01.007.

Qi, X. and Reiss, S. P. Leveraging syntax-related code for automated program repair, 2017. URL https://doi.org/10.1109/ase.2017.8115676.

Qi, Y., Mao, X., Lei, Y., Dai, Z., and Wang, C. The strength of random search on automated program repair, 2014. URL https://doi.org/10.1145/2568225.2568254.

Qian, Y., Clune, J., Barrett, C., and Avigad, J. Lean-Auto: An Interface Between Lean 4 and Automated Theorem Provers, 2025. URL https://doi.org/10.1007/978-3-031-98682-6_10.

Rubin, D. B. Formal mode of statistical inference for causal effects, 1990. URL https://doi.org/10.1016/0378-3758(90)90077-8.

Rushby, J. A formally verified algorithm for clock synchronization under a hybrid fault model, 1994. URL https://doi.org/10.1145/197917.198115.

Samoylov, E. and Vosoughi, S. Modeling Tactics as Operators: Effect-Grounded Representations for Lean Theorem Proving, 2025. URL https://doi.org/10.18653/v1/2025.mathnlp-main.12.

Shen, S., Cheng, W., Ma, M., Turcan, A., Zhang, M. J., and Ma, J. SKILLFOUNDRY: Building Self-Evolving Agent Skill Libraries from Heterogeneous Scientific Resources, 2026. URL https://doi.org/10.48550/arxiv.2604.03964.

Smith, E. K., Barr, E. T., Goues, C. L., and Brun, Y. Is the cure worse than the disease? overfitting in automated program repair, 2015. URL https://doi.org/10.1145/2786805.2786825.

Son, G., Hong, J.-W., Fan, H., Nam, H., Ko, H., Lim, S., Song, J., Choi, J., Paulo, G., Yu, Y., and Biderman, S. When AI Co-Scientists Fail: SPOT-a Benchmark for Automated Verification of Scientific Research, 2025. URL https://doi.org/10.48550/arxiv.2505.11855.

Stalmeijer, R. E., Dolmans, D., Wolfhagen, I. H. A. P., and Scherpbier, A. Cognitive apprenticeship in clinical practice: can it stimulate learning in the opinion of students?, 2008. URL https://doi.org/10.1007/s10459-008-9136-0.

Stalmeijer, R. E., Dolmans, D., Snellen-Balendong, H., van Santen-Hoeufft, M., Wolfhagen, I. H. A. P., and Scherpbier, A. J. Clinical Teaching Based on Principles of Cognitive Apprenticeship, 2013. URL https://doi.org/10.1097/acm.0b013e31828fff12.

Staples, M., Jeffery, R., Andronick, J., Murray, T., Klein, G., and Kolanski, R. Productivity for proof engineering, 2014. URL https://doi.org/10.1145/2652524.2652551.

Szegedy, C. A Promising Path Towards Autoformalization and General Artificial Intelligence, 2020. URL https://doi.org/10.1007/978-3-030-53518-6_1.

Tang, X. A Comprehensive Survey of the Lean 4 Theorem Prover: Architecture, Applications, and Advances, 2025. URL https://arxiv.org/abs/2501.18639v1.

Toyonaga, K., Nakao, M., and Watanabe, Y. Verified numerical computations for multiple and nearly multiple eigenvalues of elliptic operators, 2002. URL https://doi.org/10.1016/s0377-0427(02)00431-4.

Walter, W. V. ACRITH-XSC A Fortranlike Language for Verified Scientific Computing, 1993. URL https://doi.org/10.1016/s0076-5392(08)62841-6.

Wang, M., Tang, Y., Wang, J., and Deng, J. Premise Selection for Theorem Proving by Deep Graph Embedding, 2017. URL https://arxiv.org/abs/1709.09994v1.

Wang, Q., Kaliszyk, C., and Urban, J. First Experiments with Neural Translation of Informal to Formal Mathematics, 2018. URL https://doi.org/10.1007/978-3-319-96812-4_22.

Wang, Q., Brown, C. E., Kaliszyk, C., and Urban, J. Exploration of neural machine translation in autoformalization of mathematics in Mizar, 2020. URL https://doi.org/10.1145/3372885.3373827.

Wen, M., Chen, J., Wu, R., Hao, D., and Cheung, S.-C. Context-aware patch generation for better automated program repair, 2018. URL https://doi.org/10.1145/3180155.3180233.

Whiteside, I., Aspinall, D., Dixon, L., and Grov, G. Towards Formal Proof Script Refactoring, 2011. URL https://doi.org/10.1007/978-3-642-22673-1_18.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with Large Language Models, 2022. URL https://doi.org/10.48550/arxiv.2205.12615.

Xia, C. S., Wei, Y., and Zhang, L. Automated Program Repair in the Era of Large Pre-trained Language Models, 2023. URL https://doi.org/10.1109/icse48619.2023.00129.

Yang, K. LeanDojo Benchmark, 2023. URL https://doi.org/10.5281/zenodo.8016385.

Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models, 2023. URL https://doi.org/10.48550/arxiv.2306.15626.

Zhang, L., Quan, X., and Freitas, A. Consistent Autoformalization for Constructing Mathematical Libraries, 2024. URL https://doi.org/10.18653/v1/2024.emnlp-main.233.

Zhang, Q., Fang, C., Ma, Y., Sun, W., and Chen, Z. A Survey of Learning-based Automated Program Repair, 2023. URL https://doi.org/10.1145/3631974.

Zhou, J., Staats, C., Li, W., Szegedy, C., Weinberger, K. Q., and Wu, Y. Don't Trust: Verify – Grounding LLM Quantitative Reasoning with Autoformalization, 2024a. URL https://doi.org/10.48550/arxiv.2403.18120.

Zhou, W., Ou, Y., Ding, S., Li, L., Wu, J., Wang, T., Chen, J., Wang, S., Xu, X., Zhang, N., Chen, H., and Jiang, Y. E. Symbolic Learning Enables Self-Evolving Agents, 2024b. URL https://doi.org/10.48550/arxiv.2406.18532.

Zhu, Q., Sun, Z., an Xiao, Y., Zhang, W., Yuan, K., Xiong, Y., and Zhang, L. A syntax-guided edit decoder for neural program repair, 2021. URL https://doi.org/10.1145/3468264.3468544.

Zuo, X., Zhang, Y., Wang, H., Cai, Y., Hou, Z., Sun, J., and Dong, J. S. PAT-Agent: Autoformalization for Model Checking, 2025. URL https://doi.org/10.1109/ase63991.2025.00176.

A. Corpus Audit and Experimental Protocol

The released artifact bundle contains the project JSON files, the core report, the extended literature map, all generated figures, CSV summaries, and the script `experiments/run_experiments.py`. The experiments use no external APIs and no private data. Records were loaded from the seven top-cluster JSON files and three broad JSON corpora, normalized by ASCII title, and deduplicated by normalized title. When duplicate titles appeared, the script preferred top-cluster annotations over broad corpus metadata and then higher relevance score.

Table 8. Corpus counts used by the experiments.

Subset	Raw / deduplicated count	Notes
All JSON records	1,406 raw	includes broad and top-cluster files
Title-deduplicated records	987 dedup.	after metadata-artifact filtering
Valid-year records	979 dedup.	year between 1980 and 2026
Top-cluster records	450 dedup.	seven top-75 files after deduplication
Formal reasoning cluster	55 top records	Lean, ATP, proof search
Autoformalization cluster	55 top records	NL-to-formal and alignment
Proof engineering cluster	69 top records	ITP practice and proof maintenance
Beyond-math formalization cluster	72 top records	V&V, scientific computing, domains
Self-evolving agents cluster	56 top records	memory, reflection, scientific agents
Apprenticeship cluster	72 top records	cognitive apprenticeship and pedagogy
Repair/failure cluster	71 top records	program/proof repair and counterexamples

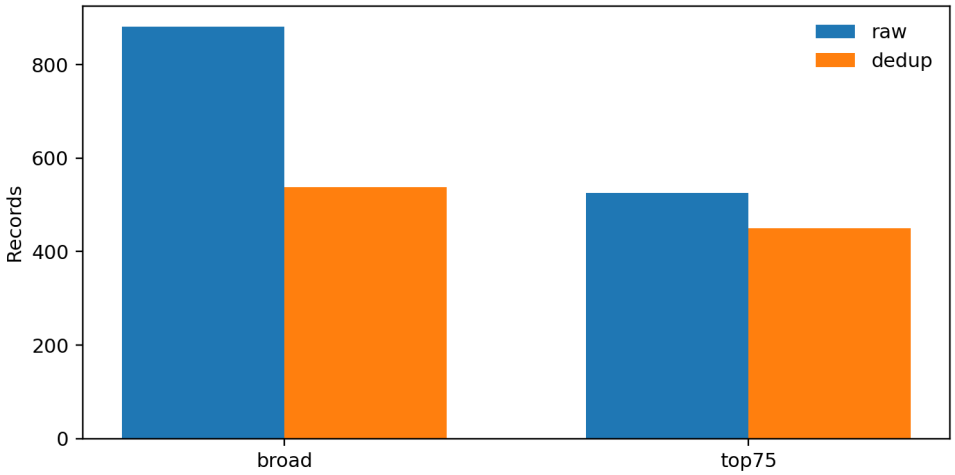

Figure 6. Raw versus title-deduplicated corpus records.

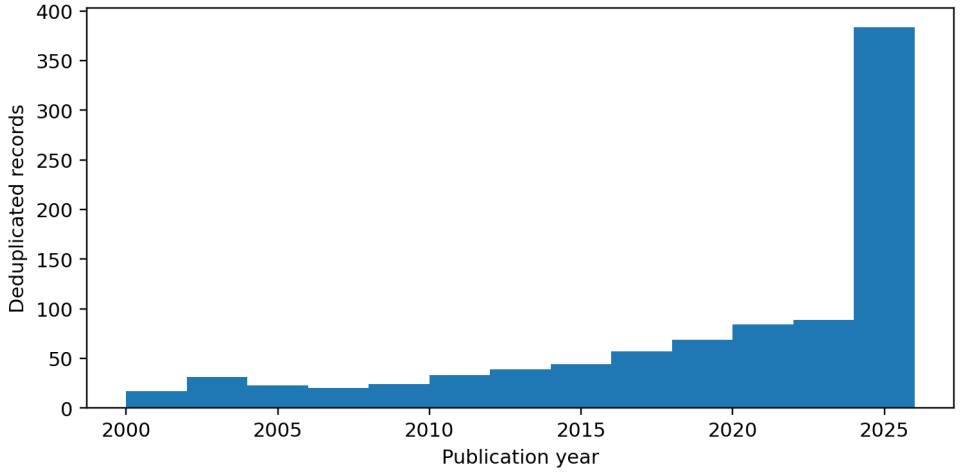

Figure 7. Year distribution of deduplicated valid-year records. Recent agent and formal-reasoning work is strongly represented, while apprenticeship and proof-engineering literatures provide historical depth.

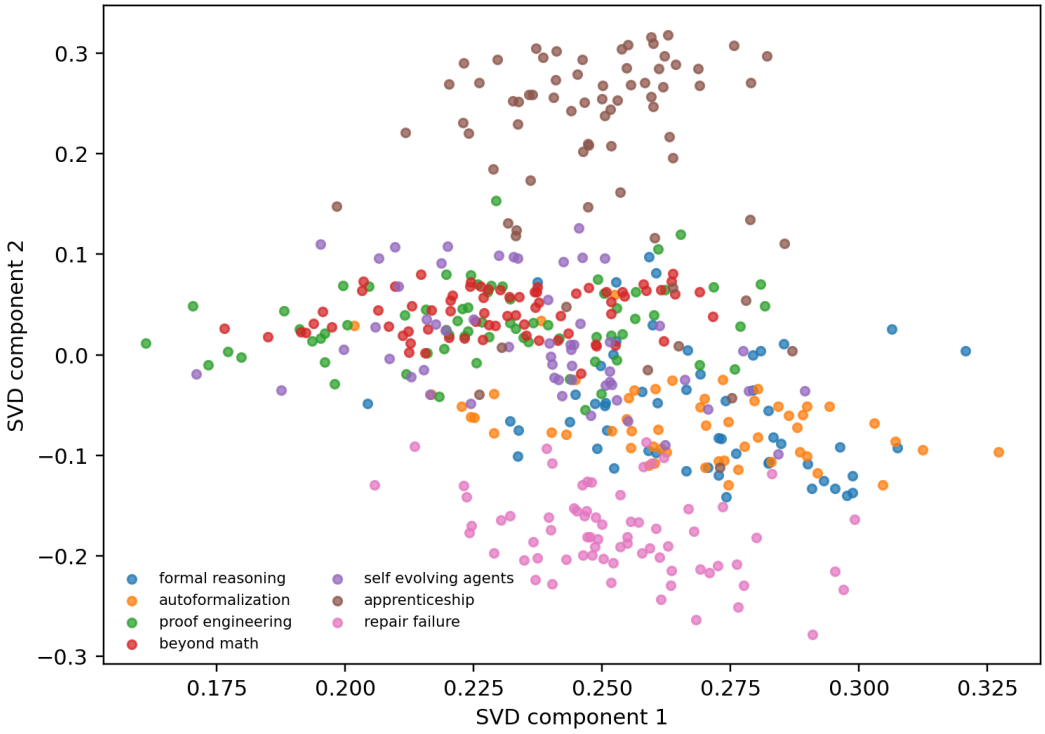

Figure 8. SVD projection of TF-IDF document vectors for the deduplicated top-cluster corpus. This is a diagnostic visualization rather than a claim of ground-truth taxonomy.

B. Additional Corpus Results

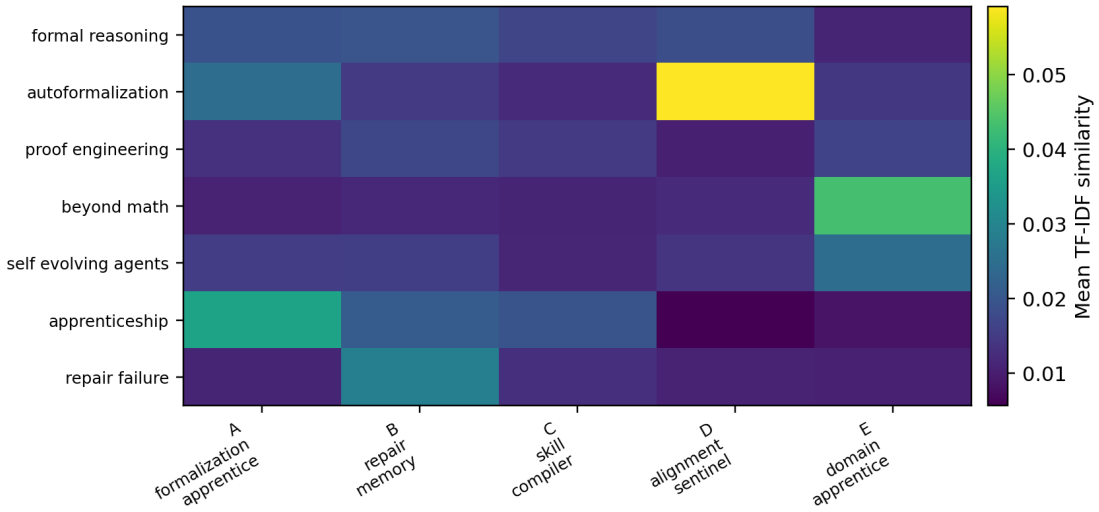

Figure 9. Architecture-family similarity. Repair memory is concentrated in repair/failure and proof-engineering records; domain apprenticeship is strongest in beyond-math formalization and scientific-agent records.

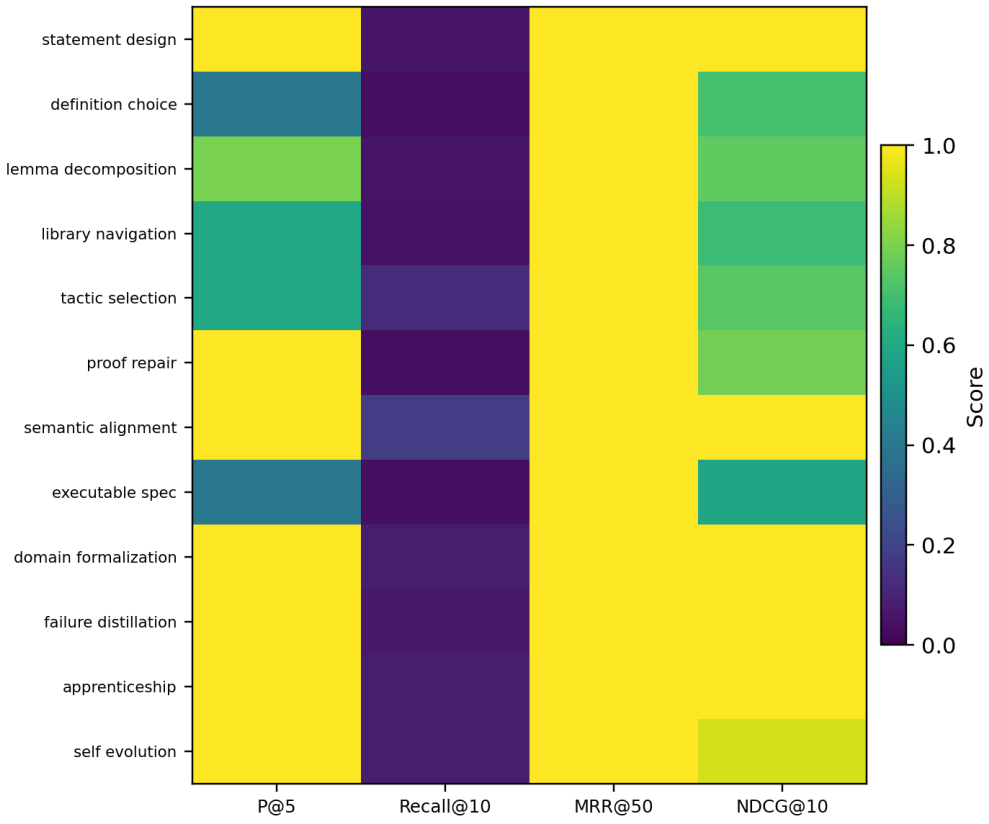

Figure 10. Per-query retrieval metrics for the full-annotation index. The weakest queries mark where craft terms are diffuse and need more explicit trace fields.

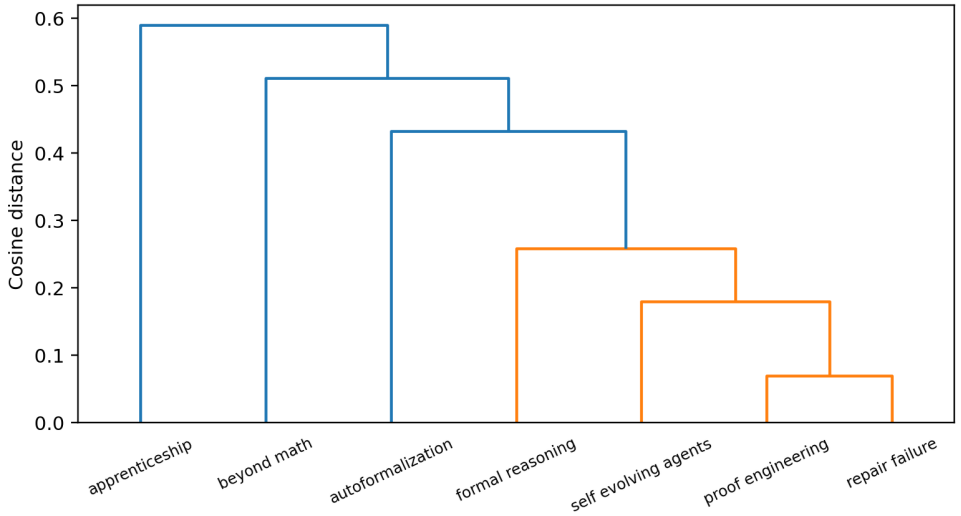

Figure 11. Cluster dendrogram using craft-similarity profiles. The tree summarizes which literatures share craft signatures, not which papers should be merged.

Topic	Top terms
1	lean, theorem, proving, prover, theorem proving, search, interactive, proofs
2	cognitive, cognitive apprenticeship, apprenticeship, students, teaching, expert, skills, education
3	program, program repair, apr, patches, bugs, test, automated program, code
4	algorithm, algorithms, engineering, safety, formal verification, systems, algorithmic, distributed
5	autoformalization, language, llms, natural language, natural, semantic, models, llm
6	ai, discovery, scientists, research, scientist, ai scientists, autonomous, agent
7	causal, inference, causal inference, effects, assumptions, causal reasoning, statistical, causality
8	numerical, verified, computation, computing, verified numerical, scientific computing, computations, bounds

Figure 12. Eight NMF topics learned from the top-cluster text. The topics are used only as interpretability aids.

Craft	Top-matching paper	Cluster	Score
statement design	ProofNet: Autoformalizing and Formally Proving Undergraduate-Level Mathematics	autoformalization	0.20
definition choice	Granger Causality and Transfer Entropy Are Equivalent for Gaussian Variables	beyond math	0.11
lemma decomposition	HybridProver: Augmenting Theorem Proving with LLM-Driven Proof Synthesis and Refinement	formal reasoning	0.15
library navigation	Mathlib4 Theorem Dependency Graph (LeanDojo Benchmark 4, v10)	formal reasoning	0.23
tactic selection	Modeling Tactics as Operators: Effect-Grounded Representations for Lean Theorem Proving	formal reasoning	0.17
proof repair	Towards Formal Proof Script Refactoring	proof engineering	0.25
semantic alignment	Improving Lean4 Autoformalization via Cycle Consistency Fine-tuning	autoformalization	0.17
executable spec	Formal Modeling and Verification of a Federated Byzantine Agreement Algorithm for Blockchain Platforms	beyond math	0.13
domain formalization	C++ Toolbox for Verified Scientific Computing I: Basic Numerical Problems	beyond math	0.23
failure distillation	Fault localization for automated program repair: effectiveness, performance, repair correctness	repair failure	0.18
apprenticeship	Application of the cognitive apprenticeship framework to a middle school robotics camp	apprenticeship	0.28
self evolution	Continual lifelong learning with neural networks: A review	self evolving agents	0.21

Figure 13. Top-matching paper for each craft query under the TF-IDF probe. These are seed exemplars for constructing skill-specific reading lists.

C. Representation Schema

A TVA system logs not just artifacts but the transformations that produced them. Each object below should be serializable, versioned, and linked to provenance.

Table 9. Full representation schema.

Object	Fields	Purpose
Goal	id, informal statement, domain, risk, desired artifact, examples	anchors all alignment and transfer checks
Informal statement	source text, assumptions, notation, examples, non-examples	prevents proving a nearby but wrong claim
Formal statement	prover/model language, imports, declarations, theorem/spec text	executable target for checking
Intended semantics	paraphrase, tests, invariants, counterexamples, witnesses	semantic-alignment oracle bundle
Proof state	goals, hypotheses, metavariables, local context, open obligations	state on which tactics operate
Library context	imports, candidate premises, unavailable premises, dependency graph	makes library navigation observable
Tactic trace	action, pre-state, post-state, used premises, latency, error	state-transformational tactic evidence
Failure signature	diagnostic, failing object, witness, likely cause, confidence	retrieval key for repair memory
Repair action	edit script, rationale, operator, validator result, delta risk	reusable repair example
Human critique	comment, correction, severity, accepted/rejected status	apprenticeship feedback signal
Provenance	source, data version, theorem/library version, toolchain hash	audit and leakage control
Transfer boundary	required preconditions, excluded domains, known brittle cases	prevents unsafe memory reuse
Confidence	syntactic, semantic, domain, calibration, abstention reason	supports calibrated deployment

D. Failure Matrix

The following matrix is the controlled failure-memory dataset used in Experiment 3. It doubles as a checklist for future benchmark construction.

Table 10. Failure matrix for tacit verification apprenticeship.

Failure	Domain	Tacit cause	Signature	Repair operator
misformalized quantifier	math	semantic alignment	quantifier order changed; theorem checks but weakens intent	statement alignment repair
missing hypothesis	math	statement design	proof state needs side condition absent from formal statement	statement alignment repair
overstrong conclusion	math	statement design	counterexample found for generated theorem	statement alignment repair
semantic back-translation gap	math	semantic alignment	formal-to-NL paraphrase fails critical condition	statement alignment repair
bad definition choice	math	definition choice	chosen representation blocks induction and library lemmas	representation refactor
typeclass resolution loop	math	definition choice	elaboration times out due to implicit instance search	representation refactor
physics dimension error	physics	domain formalization	unit analysis reveals incompatible equation terms	representation refactor

Failure	Domain	Tacit cause	Signature	Repair operator
bio ontology mismatch	bioinfo.	domain formalization	gene/protein/pathway ID mismatch	representation refactor
missing auxiliary lemma	math	lemma decomposition	main proof stalls with repeated subgoal pattern	retrieve or decompose
library blind search	math	library navigation	reproves theorem already in library; import missing	retrieve or decompose
proof state misread	math	tactic selection	tactic targets wrong goal or hypothesis name	retrieve or decompose
sorry masked failure	math	failure distillation	placeholder hides unsolved proof obligation	retrieve or decompose
brittle tactic sequence	math	tactic selection	proof works before minor simplifier or version change	proof script repair
version import break	math	proof repair	namespace or theorem name changed across library version	proof script repair
repair overfits oracle	general	proof repair	candidate passes available tests but violates independent oracle	proof script repair
causal graph mismatch	causality	domain formalization	DAG omits confounder required by identification	domain constraint repair
CI constraint inconsistency	causality	executable spec	conditional independence claims violate graph axioms	domain constraint repair
finance no-arbitrage omission	finance	domain formalization	portfolio constraints permit free lunch in test scenario	domain constraint repair
rounding invariant break	finance	executable spec	AMM or price oracle violates conservation under rounding	domain constraint repair
bio statistical leakage	bioinfo.	executable spec	pipeline validates on contaminated split	domain constraint repair
numerical enclosure gap	physics	executable spec	floating point interval proof lacks enclosure bound	domain constraint repair
agent memory overtransfer	general	self evolution	repair pattern imported into incompatible domain	memory governance
confidence miscalibration	general	failure distillation	agent reports high confidence despite unstable verification	memory governance
human critique ignored	general	apprenticeship	repeats corrected mistake across episodes	memory governance
provenance loss	general	executable spec	claim cannot be traced to proof, data, or test	memory governance
skill library duplication	general	self evolution	new skill duplicates old skill with divergent preconditions	memory governance

E. Algorithms

Algorithm 2 Failure Diagnosis and Normalization

Require: Artifact A, verifier output o, goal G_t, domain D_t

1: Parse o into diagnostics, counterexamples, and failing spans
2: Localize failure to statement, definition, tactic, library, domain invariant, or oracle
3: Compare A against intended semantics using paraphrase, examples, and witnesses
4: Assign craft cause c and confidence q
5: Emit normalized signature $(c, D_t, \text{span}, \text{witness}, q)$

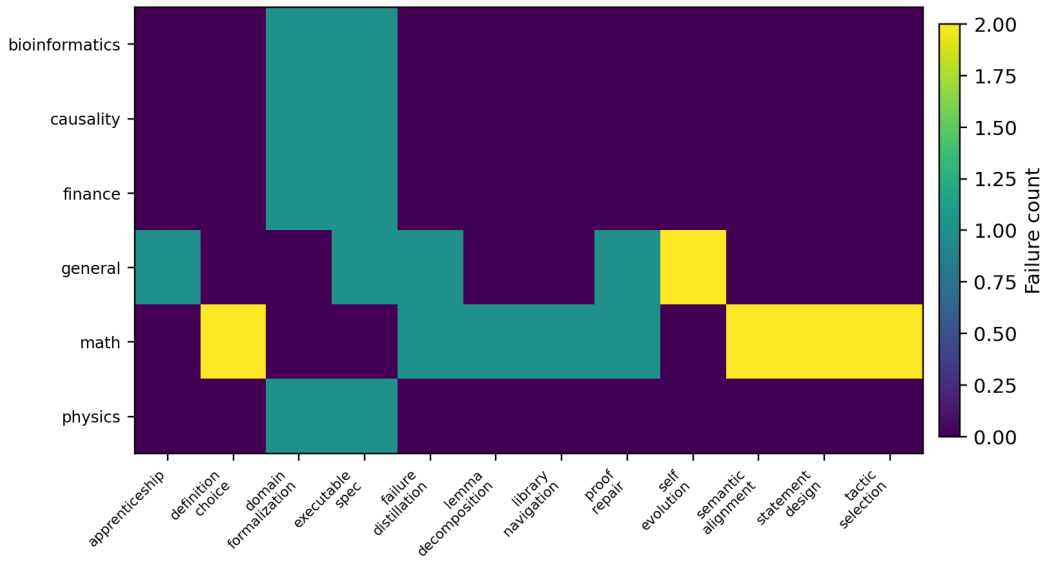

Figure 14. Domain-by-craft distribution of the failure-memory cases.

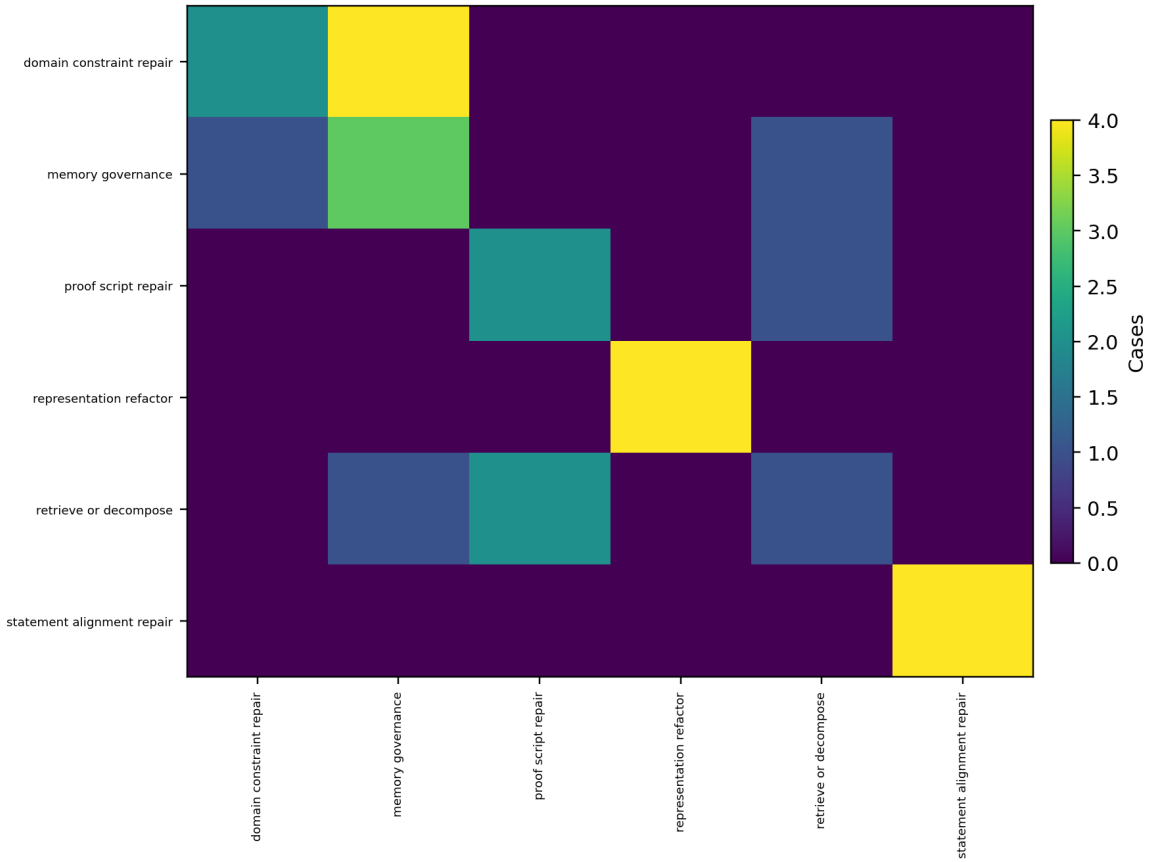

Figure 15. Confusion matrix for the transfer-gated repair-memory setting.

Algorithm 3 Repair Selection with Transfer Gating

Require: Failure signature f, memory K_t, goal G_t, domain D_t
Retrieve prior cases by failure, craft, domain, and proof-state similarity
for all candidate repair r **do**
  Estimate success, semantic faithfulness, brittleness, and validation cost
  if D_t violates stored transfer boundary of r **then**
    Downweight or reject r
  end if
end for
Return top repair plus abstention if calibrated confidence is too low

Algorithm 4 Tacit Skill Compilation

Require: Set of traces $\mathcal{T}$ with successes, failures, repairs, and critiques
Cluster traces by failure signature and repair delta
Infer preconditions from shared proof-state and domain features
Infer postconditions from validator success and semantic-alignment checks
Attach counterexamples and non-transfer cases as boundary conditions
Store skill with utility statistics, provenance, and version constraints

F. Utility Instantiation and Sentinel Details

Table 11. Concrete scoring card for the skill utility function.

Term	Operational estimate	Gate / range
p_{succ}	smoothed validation rate for same craft, domain, and oracle	$[0, 1]$
r_{repair}	reduction in repeated failure after applying the repair	$[0, 1]$
f_{sem}	minimum of facet, example, backtranslation, and domain checks	hard gate if low
v_{reuse}	held-out reuse or retrieval gain attributable to the skill	$[0, 1]$
τ_{transfer}	match between current task and stored transfer boundary	hard gate
c_{verify}	normalized verifier, checker, or human-review cost	penalty
b_{brittle}	failure rate under perturbations or version changes	penalty
$\text{risk}(D)$	consequence of accepting a wrong artifact in the domain	penalty

Default weights in the released code are deliberately conservative: semantic faithfulness and transfer validity dominate raw success probability in high-risk domains. The selector therefore prefers an abstention over a high-pass-rate repair when the repair has weak semantic evidence or an unlicensed transfer.

Algorithm 5 Semantic Alignment Sentinel

Require: Informal goal I, artifact A, domain D, validators $\mathcal{V}$
Extract facets: objects, quantifiers, assumptions, conclusion, examples, invariants
Normalize A into comparable facets plus executable or symbolic probes
Generate discriminating tests, counterexamples, or backtranslations when available
Estimate $q_{\text{facet}}, q_{\text{example}}, q_{\text{back}}, q_{\text{domain}}$
Set $f_{\text{sem}} \leftarrow \min(q_{\text{facet}}, q_{\text{example}}, q_{\text{back}}, q_{\text{domain}})$
if f_{sem} is high and no validator disagrees **then**
    accept alignment evidence and pass artifact to verifier
else if localized repair template applies **then**
    emit repair request with witness and changed facet
else
    abstain and request stronger oracle or human critique
end if

G. End-to-End Semantic-Sentinel Demonstration

The demo in `experiments/end_to_end_semantic_demo.py` represents each informal arithmetic claim by a hidden reference predicate over a finite integer domain. Candidate formalizations are executable predicates. The sentinel compares truth sets, returns counterexample witnesses for disagreement, classifies the candidate as aligned, too strong, too weak, or mixed, and applies a repair template when one is licensed by the stored failure type. This finite-domain setup is intentionally modest: it tests the architecture's control flow without claiming to solve open autoformalization.

Table 12. Semantic-sentinel demo summary.

Metric	Value
Tasks	18
Misaligned candidates	12
Sentinel precision	1.00
Sentinel recall	1.00
Repair successes	9/12
Abstentions	3

Representative cases include missing hypotheses (e.g., formalizing an implication as an unconditional statement), lost domain restrictions (e.g., omitting nonzero or positive assumptions), and reversed implication direction. In all flagged cases the witness is stored as negative evidence so that the same repair is not transferred without checking the relevant facet and domain boundary.

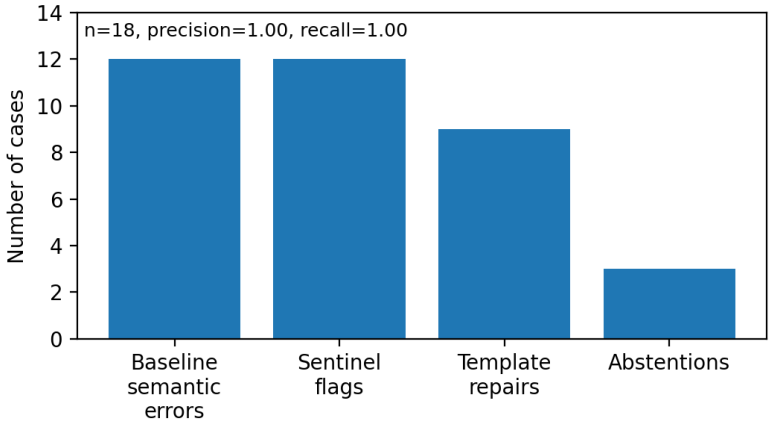

Figure 16. End-to-end semantic-sentinel demo. A syntax-only executable baseline accepts all candidates; the sentinel detects semantic misalignment, repairs template-covered cases, and abstains when the repair library lacks an applicable operator.

H. Benchmark Design

ApprenticeshipBench-SESA should be distributed as a suite of small but process-rich tasks. Each task instance contains an informal claim, a target formal or executable artifact, a hidden alignment or validation oracle, versioned toolchain information, and at least one failure trace. The task is scored over episodes, not only final answers, because the object of measurement is apprenticeship.

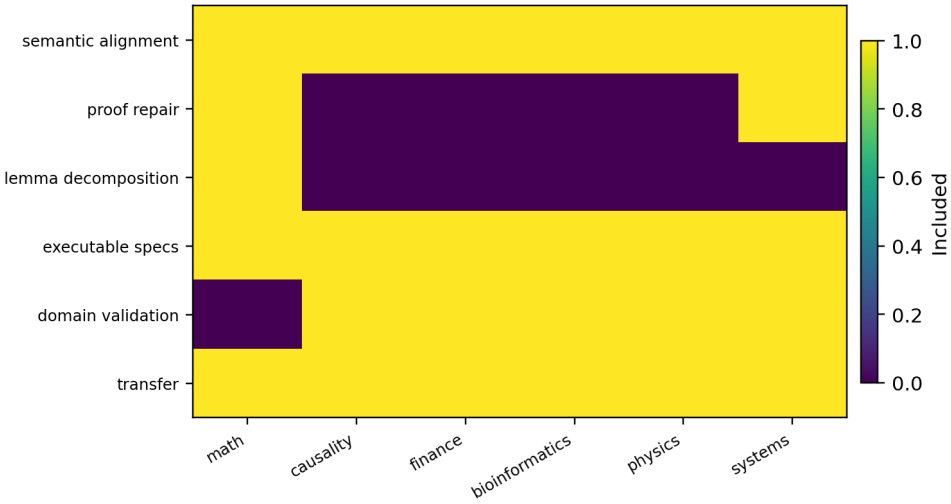

Figure 17. Coverage map for proposed benchmark task families and domains.

I. Tacit Craft Taxonomy

Table 13. Proposed benchmark metrics.

Metric	Definition
Repair Success	fraction of failures repaired and revalidated under the correct oracle
Semantic Faithfulness	agreement between informal intent and formal/executable artifact
Lemma Reuse	verified reuse of compiled intermediate claims in later tasks
Brittleness Reduction	survival under library versions, notation changes, and perturbations
Human Correction Efficiency	reduction in human edits or critiques per solved task
Cross-Domain Transfer	successful reuse under declared transfer boundaries
Skill Improvement Slope	rate of improvement across repeated apprenticeship episodes

Table 14. Expanded craft taxonomy with transfer limits.

Craft	Internal representation	Common failure	Transfer limit
Statement design	theorem templates, quantifier graphs	under/over-stated claim	depends on logic and domain assumptions
Definition choice	representation families, equivalences	unusable induction or coercions	rarely transfers without domain ontology
Lemma decomposition	proof DAG and subgoal motifs	monolithic proof search	transfers within similar proof topology
Library navigation	dependency graph and premise embeddings	missing known theorem	version and namespace sensitive
Tactic selection	state-delta operators	brittle or wrong tactic	prover-specific semantics
Proof repair	failure-to-patch cases	overfitted patch	oracle-strength dependent
Semantic alignment	NL-formal diff and examples	provable wrong theorem	domain-intent dependent
Executable spec	tests, model-check witnesses	shallow validation	constrained by test coverage
Domain formalization	causal/finance/bio/physics invariants	wrong scientific model	domain expertise required
Failure distillation	normalized signatures and lessons	repeated mistake	depends on logging quality
Apprenticeship	critique history and scaffolds	ignored feedback	human-process dependent
Self-evolution	skill utility and memory policy	unsafe overtransfer	requires calibrated boundaries

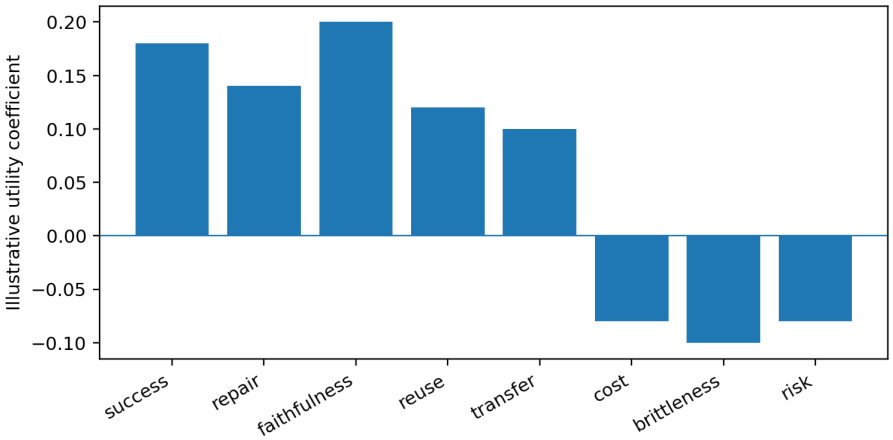

Figure 18. Illustrative skill-utility coefficients used in the formalism. Positive terms reward success, repair value, faithfulness, reuse, and transfer; negative terms penalize verification cost, brittleness, and domain risk.

J. Cross-Disciplinary Atlas

The atlas below maps literatures to mechanisms and agent modules. It is used to ensure that TVA is not reduced to theorem proving alone.

Table 15. Cross-disciplinary atlas.

Literature	Mechanism imported	Agent module	Pitfall
Interactive theorem proving	proof states, tactics, elaboration feedback	apprentice, verifier	syntax success mistaken for intent
Lean/mathlib ecosystems	imports, dependencies, library search	library memory	version brittleness
Autoformalization	NL-to-formal translation and back-translation	apprentice, sentinel	semantic illusion
Semantic alignment	misalignment taxonomies and selection tests	sentinel	insufficient behavioral oracle
Proof engineering	proof maintenance, refactoring, CI	repair memory	ignoring process traces
Program repair	fault localization, patch ranking, validation	repair memory	test overfitting
Model checking	counterexamples and temporal specs	executable spec learner	state-space mismatch
Scientific V&V	verification versus validation distinction	domain apprentice	validating wrong model class
Causal inference	DAGs, potential outcomes, CI constraints	domain apprentice	hidden confounding
Finance formalization	no-arbitrage and invariant repair	domain apprentice	incomplete market assumptions
Bioinformatics workflows	ontology alignment and leakage checks	domain apprentice	identifier drift
Physics formalization	units, dimensions, numerical enclosures	domain apprentice	informal approximations
Cognitive apprenticeship	modeling, coaching, scaffolding, fading	training regime	invisible expert reasoning
Proof pedagogy	goal decomposition and mathematical language feedback	apprentice	toy-task overfitting
Case-based reasoning	retrieve/adapt/store episodes	repair memory	shallow similarity
Deliberate practice	targeted failure drills	curriculum	narrow drills lacking transfer
HCI for proof assistants	proof-state legibility and mixed initiative	interface	hiding uncertainty
After-action review	near-miss capture and lessons learned	failure distillation	root-cause seduction
Software engineering	refactoring, tests, CI, code review	skill compiler	treating proofs as static text
Continual learning	retention, replay, transfer, forgetting	memory policy	catastrophic forgetting
Agent reflection	self-critique and memory updates	self-evolution	ungrounded introspection
Scientific agents	hypothesis, code, data, literature loops	domain apprentice	unsupported claims

K. Additional Implementation Notes

A practical implementation should log both successful and failed proof-state transitions, preserve all verifier outputs, maintain a case base of rejected repairs, and explicitly label transfer boundaries. The memory store should expose three retrieval interfaces: exact symbolic retrieval over theorem names and dependencies, semantic retrieval over informal statements and

rationales, and failure retrieval over normalized diagnostics. The repair selector should be conservative in high-risk domains: when semantic faithfulness or domain validation is weak, abstention is a valid outcome.

L. Extended Related Work

Formal reasoning systems supply the executable substrate for the apprenticeship loop: Lean and Lean 4 (de Moura et al., 2015a; de Moura & Ullrich, 2021), connection-based and ATP-assisted proving (Otten & Bibel, 2003; Otten, 2008; Qian et al., 2025), retrieval and premise selection (Wang et al., 2017; Yang et al., 2023), proof-progress and tactic-effect representations (George et al., 2025; Samoylov & Vosoughi, 2025). Autoformalization and semantic alignment motivate the sentinel: neural translation (Wang et al., 2018; 2020), LLM autoformalization (Wu et al., 2022; Jiang et al., 2023a), ProofNet (Azerbayev et al., 2023), consistency and syntax-feedback loops (Zhang et al., 2024), PAT-Agent (Zuo et al., 2025), FormalAlign (Lu et al., 2024), and autoformalized verification of quantitative reasoning (Zhou et al., 2024a). Proof engineering and scientific formalization broaden the scope beyond isolated theorems (Beckert et al., 2015; Klein, 2014; Staples et al., 2014; Dietrich et al., 2013; Malecha, 2015; Oberkampf & Roy, 2010; Lincoln & Rushby, 2005; Hurd, 2021; Palmer et al., 2006). Self-evolving agents and scientific workflow agents provide memory, reflection, and long-horizon evaluation frames (Gottweis et al., 2025; Liang et al., 2025; Son et al., 2025; Mitchener et al., 2025; Zhou et al., 2024b; Chen et al., 2024; Fang et al., 2025). Cognitive apprenticeship provides the conceptual reason to preserve expert traces and feedback (Collins, 1988; Collins et al., 1988; Brown et al., 1989; Collins et al., 1991; 2018). Repair and failure-learning provide the generate-diagnose-patch-validate loop (Goues et al., 2019; Qi et al., 2014; Wen et al., 2018; Smith et al., 2015; Xia et al., 2023; Jiang et al., 2023b).

M. Future Work and Open Problems

Table 16. Research directions enabled by the TVA agenda.

Direction	One-line contribution
Tacit Verification Apprenticeship	general architecture for learning verification craft from traces
Repair Memory for Proof Assistants	case-based memory for proof and specification failures
Semantic Alignment Sentinel	detecting provable but misaligned formal statements
Skill Compiler for Formalization	converting failed attempts into typed reusable skills
Domain Verification Apprentice	domain-specific validators for causality, finance, bio, and physics
ApprenticeshipBench-SESA	benchmark suite for skill improvement over episodes
Failure-Distillation Curriculum	deliberate-practice schedule built from near misses
Transfer-Gated Scientific Agents	memory governance model for safe cross-domain reuse

Table 17. Ten open problems.

No.	Problem
1	How should semantic faithfulness be measured when full informal-formal equivalence is unavailable?
2	Which proof-state features best predict that a local tactic sequence will transfer?
3	How can agents learn definition-choice craft without requiring expensive expert demonstrations?
4	What is the right granularity for compiled verification skills: tactic, lemma, proof plan, or domain workflow?
5	How should repair memories represent failed repairs and near misses?
6	How can benchmark designers prevent proof-repair and program-repair systems from overfitting shallow oracles?
7	How should human critiques be transformed into durable agent memory without amplifying one expert's idiosyncrasies?
8	What domain validators are adequate for causal, finance, bioinfo., and physics settings?
9	How can agents detect when a proof is correct but the scientific model is inadequate?
10	What governance policy should delete, quarantine, or downgrade a learned skill after a version or domain shift?