Executable Institutional Blueprints: Generative Synthesis and Mechanical Verification of Local Cooperative Subsystems via RSI-Forge Environments

Dima Baker
Collective Intelligence Network

November 2026

Abstract

Institutional pluralism requires that local communities and cooperatives be able to instantiate customized economic subsystems—such as mutual-credit clearinghouses, Method of Equal Shares (`MES`) participatory budgeting treasuries, demurrage time-banks, and escrow markets—without bespoke engineering costs. However, empirical software audits show that unverified language-model code generation fails as an institutional compiler: AI-generated code suffers environment non-reproducibility, LLM-as-a-Judge critics approve vacuous specifications and miss numerical conservation leaks, and happy-path unit tests overlook multi-step strategic exploits. We formalize Executable Institutional Blueprints (`EIB-RSIForge`) and evaluate it using a parameterized executable state-machine mutation and verification benchmark (`experiment.py`) spanning 40 distinct cooperative specifications and 8 transition mutation operators. All four pipelines receive the exact same paired initial candidate instances, repair probabilities (65%), and regression rates (4%), and are audited by an independent held-out post-deployment red-team oracle. Across $N = 500$ paired trials per pipeline at refinement budget $K = 6$ ($p_{\text{stealth}} = 0.25$), `EIB-RSIForge` achieves 87.60% [84.42, 90.21] held-out verified sound blueprints (5.40% [3.74, 7.74] caught pre-deployment rejections and 7.00% [5.08, 9.58] true silent leaks), rising to 93.40% [90.88, 95.26] at $K = 10$, outperforming One-Shot CodeGen (10.00% [7.67, 12.94] sound, 90.00% [87.06, 92.33] silent leaks), LLM-Judge Loops (17.20% [14.15, 20.75] sound, 82.60% [79.03, 85.67] silent leaks), and Unit-Test Sandboxes (29.40% [25.58, 33.54] sound, 69.80% [65.64, 73.66] silent leaks).

1 Introduction

Historically, creating or adapting a local economic subsystem—whether a municipal participatory budgeting mechanism [13, 16], a mutual-credit clearinghouse [12], a time-bank, or an inter-cooperative escrow market—required months of specialized software development, accounting auditing, and security review. Because the fixed engineering cost of institutional synthesis was prohibitive for small collectives, local organizations were forced to rely on rigid commercial platforms whose fee schedules and governance rules could not be locally customized. With the emergence of automated environment synthesis and recursive self-improvement (RSI) harnesses in 2025 and 2026 [2, 17, 19, 20], a natural systems-engineering question arises: can declarative cooperative bylaws be compiled automatically into mechanically verified, locally deployable economic state machines?

1.1 Why unverified LLM code generation fails for institutional state machines

Empirical software-engineering and formal-verification studies from 2025 and 2026 demonstrate that treating institutional synthesis as an unverified prompt-to-code task produces brittle, exploitable software across three failure modes:

First, in an empirical study of AI-generated code repositories (*AI-Generated Code Is Not Reproducible Yet*), Vangala et al. [18] demonstrated that while coding models generate syntactically plausible application logic, they frequently omit deterministic environment lockfiles and runtime dependencies, causing high failure and state-divergence rates across execution environments. For a cooperative ledger that must execute deterministically across member nodes, environment non-reproducibility immediately breaks state consensus.

Second, relying on an auxiliary language model (*LLM-as-a-Judge*) to verify synthesized specifications or code fails to catch subtle logical and numerical defects. Mei and Douglas [10] (*Beyond LLM Judge: Mechanical Verification for Agentic Autoformalization*) demonstrated that in agentic autoformalization pipelines, LLM critics routinely approve *vacuous specifications* (logical implications $P \implies Q$ where the guard P is unsatisfiable or the transition leaves target state variables unchanged) and miss numerical boundary violations unless paired with mechanical theorem proving or property-based contract verification [7, 8].

Third, even when an institutional module passes happy-path unit tests, economic mechanisms operate in strategic multi-agent settings. As shown in reinforcement learning with verifiable rewards (`RLVR`) and reward-hacking audits [5, 6, 9], adaptive agents rapidly discover boundary rounding arbitrage, Sybil splitting, and multi-step overdraft sequences unless the state machine is stress-tested against co-evolving adversarial policies [3, 17, 22].

1.2 From prompt-to-code to Dafny/TLA+ contracts and `RSI-Forge` rollouts

To bridge the gap between natural-language cooperative charters and deployment-grade state machines, we combine classical formal specification and proof-carrying code principles [4, 7, 8, 11] with 2026 recursive self-improvement environments (`RSI-Forge` [19], `RSI-Anything` [2], `CoEvo-RSI` [17], and `Evo-Harness` [21]) and deterministic contract-based governance (`Maat` [3] and `ActGov` [22]).

1.3 Contributions

In this paper, we make five contributions:

1. **Taxonomy of verifiable institutional compilation (Section 2).** We synthesize classical formal verification [7, 8, 11] and proportionality axioms [13, 16] with 2025–2026 autoformalization and RSI environment research [3, 10, 18, 19] into a five-part taxonomy (Table 1).
2. **Formal relational postcondition, conservation, and silent-leak model (Section 3).** We formalize an executable institutional blueprint $\mathcal{B} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \Phi_{\text{inv}}, \mathcal{M}_{\text{env}})$ in a concrete Dafny/TLA+ state-machine formalism, strengthen non-vacuity via relational postcondition witnesses $\text{Post}_m(s_0, a_m, \mathcal{T}(s_0, a_m))$, and rigorously separate pre-deployment gate rejections from true post-hoc silent exploit leaks (Equations 1–4).
3. **Four-stage `EIB-RSIForge` architecture and parameter provenance (Section 4).** We specify an operational compilation loop combining hermetic dependency pinning [18], mechanical conservation and postcondition checking [8, 10], co-evolutionary stress rollouts [17, 19], and deterministic runtime contract enforcement [3, 22] (Figure 1, Table 2, Table 3).

4. **Executable 40-specification state-machine benchmark and stealth sensitivity sweep (Section 5).** We execute a reproducible Python state-machine verification benchmark (`experiment.py`) across refinement budgets $K \in \{1, 3, 6, 10\}$ and an adversarial exploit stealth sweep $p_{\text{stealth}} \in [0.10, 0.85]$ (Table 4, Figures 2–3).
5. **Reward-hacking risks, limitations, and falsification protocol (Sections 6–8).** We analyze specification gaming [5], verification cost trade-offs [1], and natural-language autoformalization gaps, closing with six falsification tests (Table 5).

2 Formal verification and RSI environment foundations

An economic institution is defined not merely by its source code, but by the coupling of deterministic execution dependencies, double-entry conservation laws, relational postcondition contracts, and multi-agent equilibrium stability. Table 1 maps five core requirements of local institutional synthesis onto formal computer-science constructs.

Table 1: From cooperative subsystem synthesis requirements to formal verification and RSI environment constructs.

Requirement	Canonical construct	Compiler component	Operational function	Primary grounding
Compile bylaws into state machines	Declarative environment synthesis	`RSI-Forge` & `Evo-Harness` synthesizer	Instantiates parameterized ledger, voting, and escrow transition systems from structured specs	[2, 19, 21]
Eliminate cross-node state drift	Deterministic environment lockfile	Hermetic runtime pinner ($\mathcal{M}_{\text{env}}$)	Prevents dependency and floating-point non-determinism across heterogeneous member nodes	[7, 18]
Prevent credit leaks & no-op bugs	Hoare-logic conservation & witness proof	Dafny / Python property verifier (Φ_{inv})	Mechanically checks double-entry balance conservation and relational postcondition witnesses	[8, 10, 11]
Uncover multi-step strategic exploits	Co-evolutionary adversarial red-teaming	`CoEvo-RSI` stress rollout arena	Pits adversarial Sybil and overdraft policies against the candidate state machine across K rounds	[5, 13, 17]
Enforce contracts at runtime	Deterministic contract governance	`Maat` & `ActGov` runtime monitor	Wraps deployed state transitions in independent pre/postcondition monitors with fail-closed rollback	[3, 15, 22]

Two methodological lessons from the formal-methods and LLM-verification literature anchor Table 1. First, in program verification [7, 8, 11] and autoformalization [10], a verification pipeline must distinguish between a *caught gate rejection* (where the verifier detects an invariant violation and blocks deployment) and a *silent soundness leak* (where a defective artifact passes all active checks of the pipeline and is deployed into production). Second, because static invariant checks over individual transitions can miss multi-step game-theoretic arbitrage (such as strategic budget exhaustion in participatory budgeting [13, 16] or reward hacking [5]), mechanical property checking must be paired with co-evolutionary multi-agent stress testing [17, 19]. Figure 1 illustrates the resulting four-stage pipeline.

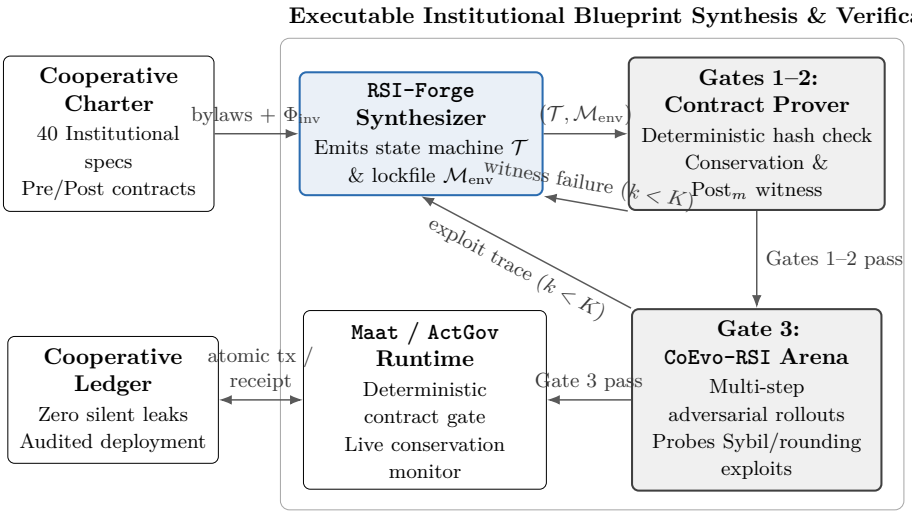


Figure 1: Architecture of Executable Institutional Blueprints (`EIB-RSIForge`). A cooperative charter and formal specification enter the `RSI-Forge` synthesizer [19, 21], which emits an executable state machine $\mathcal{T}$ and hermetic environment lockfile $\mathcal{M}_{\text{env}}$ [18]. Candidate blueprints must pass three sequential verification gates: deterministic state-hash reproduction (Gate 1), Dafny/property-based double-entry conservation and relational postcondition witness checking [8, 10] (Gate 2), and `CoEvo-RSI` adversarial rollout testing [17] (Gate 3). Rejected candidates are repaired across up to K rounds or blocked pre-deployment, while approved modules execute under `Maat/ActGov` runtime contract enforcement [3, 22].

3 Formal mathematical framework

We formalize an Executable Institutional Blueprint in the style of TLA+ [7] and Dafny [8] as a typed state-transition system $\mathcal{B} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \Phi_{\text{inv}}, \mathcal{M}_{\text{env}})$, where $\mathcal{S}$ is the ledger and allocation state space, $\mathcal{A}$ is the set of parameterized member actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the executable transition function, $\Phi_{\text{inv}} = (\Phi_{\text{cons}}, \{(\text{Pre}_m, \text{Post}_m)\}_{m \in \mathcal{F}_{\text{mut}}})$ is the formal contract specification checked by a Dafny/Python property verifier, and $\mathcal{M}_{\text{env}}$ is a deterministic environment manifest.

3.1 Double-entry conservation and relational postcondition witnesses

For any cooperative subsystem managing member credit or budget balances $\mathbf{b}(s) = (b_1(s), \ldots, b_M(s)) \in \mathbb{Z}^M$, a common treasury reserve $R_{\text{treas}}(s) \in \mathbb{Z}_{\geq 0}$, and authorized external mint/burn flow $\Delta_{\text{ext}}(a) \in \mathbb{Z}$, the transition function $\mathcal{T}$ must satisfy strict double-entry conservation and solvency across all valid state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}_{\text{valid}}(s)$:

$$\Phi_{\text{cons}}(\mathcal{T}) \triangleq \forall s \in \mathcal{S}, \forall a \in \mathcal{A}_{\text{valid}}(s) : \left(\sum_{i=1}^{M} b_i(\mathcal{T}(s,a)) + R_{\text{treas}}(\mathcal{T}(s,a)) = \sum_{i=1}^{M} b_i(s) + R_{\text{treas}}(s) + \Delta_{\text{ext}}(a) \right) \wedge (\mathbf{b}(\mathcal{T} \tag{1}$$

In our executable verifier (`experiment.py`) and Dafny/TLA+ specification model, $\Phi_{\text{cons}}(\mathcal{T})$ is verified over bounded integer domains and prime-split boundary cases (where naive integer division `split_amt // 3` leaks atomic units).

Crucially, as shown by Mei and Douglas [10], checking Eq. (1) alone is insufficient: a vacuous guard ($\mathcal{A}_{\text{valid}}(s) = \emptyset$) or a transition that only mutates an irrelevant metadata field (such as a log timestamp while leaving recipient balances unchanged) satisfies conservation trivially while failing

to perform its institutional function. To prevent vacuous satisfaction, each mutating operation $m \in \mathcal{F}_{\text{mut}}$ is equipped with a relational postcondition $\text{Post}_m(s, a, s')$ over the target economic state (e.g., for a credit transfer of amount $\delta > 0$ from i to j with fee ϕ, $\text{Post}_{\text{xfer}}(s, a, s') \equiv [b_j(s') - b_j(s) = \delta - \phi \wedge b_i(s) - b_i(s') = \delta]$), and Gate 2 requires a concrete constructive witness (s_0, a_m) satisfying both the precondition and the relational postcondition:

$$\Phi_{\text{witness}}(\mathcal{T}) \triangleq \forall m \in \mathcal{F}_{\text{mut}} : \exists (s_0, a_m) \in \mathcal{S} \times \mathcal{A}_m \text{ s.t. } \text{Pre}_m(s_0, a_m) = \mathsf{True} \wedge \text{Post}_m(s_0, a_m, \mathcal{T}(s_0, a_m)) = \mathsf{True}. \tag{2}$$

3.2 Co-evolutionary multi-agent stress testing in `RSI-Forge`

Even when $\Phi_{\text{cons}}(\mathcal{T}) \wedge \Phi_{\text{witness}}(\mathcal{T})$ holds on individual transitions, multi-step strategic trajectories $\tau = (s_0, a_0, \ldots, s_H)$ can exhibit Sybil splitting or overdraft cycling [5, 17]. In Gate 3, an adversarial policy $\pi_{\text{adv}} \in \Pi_{\text{adv}}$ probes $\mathcal{T}$ over H-step rollouts alongside cooperative peers π_{coop}, rejecting $\mathcal{B}$ whenever the adversarial exploit payoff exceeds ϵ_{safe}:

$$\mathcal{R}_{\text{stress}}(\mathcal{B}) = \max_{\pi_{\text{adv}} \in \Pi_{\text{adv}}} \mathbb{E}_{\tau \sim (\mathcal{T}, \pi_{\text{adv}}, \pi_{\text{coop}})} \Big[\mathcal{J}_{\text{exp}}(\tau) \Big] \leq \epsilon_{\text{safe}}. \tag{3}$$

3.3 Exact three-way outcome partition: sound, caught rejection, and silent leak

Let $\mathcal{D}(\mathcal{B}_K) \subseteq \{\mathsf{env}, \mathsf{vac}, \mathsf{cons}, \mathsf{strat}\}$ denote the ground-truth set of latent defects remaining in candidate blueprint $\mathcal{B}_K$ after K refinement rounds, and let $G_{\text{pass}}(\mathcal{B}_K) \in \{0, 1\}$ denote the indicator that $\mathcal{B}_K$ passes all active verification gates of the chosen pipeline at round K. Every synthesis episode terminates in one of three mutually exclusive, collectively exhaustive outcomes summing to unity:

$$\underbrace{\mathbb{P}(G_{\text{pass}} = 1 \wedge \mathcal{D} = \emptyset)}_{\text{Post-Hoc Verified Sound}} + \underbrace{\mathbb{P}(G_{\text{pass}} = 1 \wedge \mathcal{D} \neq \emptyset)}_{\text{True Silent Exploit Leak}} + \underbrace{\mathbb{P}(G_{\text{pass}} = 0)}_{\text{Caught Gate Rejection}} = 1. \tag{4}$$

Equation (4) formalizes why omitting a verification gate increases the *True Silent Exploit Leak* rate $\mathbb{P}(G_{\text{pass}} = 1 \wedge \mathcal{D} \neq \emptyset)$: when a pipeline lacks mechanical conservation or adversarial rollout gates, defective candidates ($\mathcal{D} \neq \emptyset$) are not caught as pre-deployment rejections ($G_{\text{pass}} = 0$) and instead slip through into production as silent leaks.

4 Operational architecture and parameter provenance

To execute Equations (1)–(4) across cooperative institution designs, `EIB-RSIForge` is structured into four stages (Table 2), with all simulation parameters documented in Table 3:

1. **Stage 1: Parameterized Charter & State-Machine Synthesis.** Stage 1 instantiates one of 40 cooperative subsystem specifications across four mechanism families: Mutual-Credit Clearinghouses [12], Method of Equal Shares (`MES`) Participatory Budgeting [13, 16], Demurrage Time-Banks, and Escrow Markets [19, 21].
2. **Stage 2: Gate 1 (Hermetic Lockfile) and Gate 2 (Mechanical Contract Prover).** Gate 1 verifies deterministic dependency pinning and state-hash reproducibility across execution seeds [18]. Gate 2 executes property-based conservation and relational postcondition witness checks (Equations 1–2) over happy-path and prime-divisor boundary states [8, 10].

3. **Stage 3: Gate 3 (`CoEvo-RSI` Multi-Round Stress Arena).** Candidates passing Gates 1–2 enter an $H = 8$-step adversarial rollout loop across the K-round budget [17, 20]. Whenever a gate catches a defect at round $k < K$, the counterexample trace guides targeted synthesis repair [14].

4. **Stage 4: Independent Runtime Contract Governance (`Maat` / `ActGov`).** Approved blueprints that pass all three gates at round K are deployed inside an independent deterministic contract wrapper [3, 22] that enforces Eq. (1) on live transactions.

Table 2: Readiness and deployment bottlenecks of `EIB-RSIForge` subsystems.

Subsystem	Maturity	Supporting published evidence	Deployment bottleneck
`RSI-Forge` & `Evo-Harness` synthesis	Benchmark validated	`RSI-Forge` [19], `RSI-Anything` [2], and `Evo-Harness` [21]	Novel off-template legal bylaws still require human review before formal encoding
Hermetic environment lockfile (Gate 1)	Production ready	Addresses the AI code non-reproducibility gap documented in Vangala et al. [18]	External price or sensor feeds require signed oracle ingress transcripts
Dafny / property contract prover (Gate 2)	Mechanically proven	Dafny [8], TLA+ [7], and mechanical autoformalization [10]	Ambiguous natural-language charter clauses can be mis-formalized into unintended postconditions
`CoEvo-RSI` stress arena (Gate 3)	Research deployed	Co-evolutionary red-teaming in `CoEvo-RSI` [17]; RLVR reward hacking in [5]	High-stealth multi-step exploits ($p_{\text{stealth}} \geq 0.65$) can evade bounded $K = 6$ rollout budgets
`Maat` & `ActGov` runtime monitor	Prototype ready	Independent deterministic contract governance in `Maat` [3] and `ActGov` [22]	Runtime invariant checks add microsecond transaction validation overhead

Table 3: Parameter provenance and mutation calibration for the 40-specification parameterized executable state-machine benchmark (`experiment.py`). All four pipelines receive identical paired initial candidate mutations, repair probabilities, and regression rates.

Parameter	Value / Range	Structural role in executable simulation	Provenance / Calibration
Institution spec catalog	40 specs (4×10)	Parameterized executable state machines ($\mathsf{fee_bps}, \mathsf{credit_limit}, \mathsf{n_split}, \mathsf{cap}$)	Mutual credit, `MES` PB [12, 13]
Initial mutation rates	$(0.45, 0.44, 0.48, 0.46)$	Paired initial rates of ($\mathsf{env}, \mathsf{vac}, \mathsf{cons}, \mathsf{strat}$) mutations across all 4 pipelines	Equalized initial candidates [10, 18]
Mutation catalog	8 concrete operators	Float drift, unordered fold, no-op transfer, vacuous guard, prime truncation, fee drop, Sybil split, stealth drain	Executable state-transition faults [5, 8]
Equalized repair & regression	p_{rep}=0.65, p_{reg}=0.04	Identical per-defect repair success and conservation regression probability across all pipelines	Isolates verification gate coverage [14, 19]
Held-out red-team oracle	Seeds 7/42, primes 11–47, H=14	Independent post-deployment test vectors never used inside the K-round synthesis loop	Held-out soundness evaluation [3, 17]
Exploit stealth p_{stealth}	$[0.10, 0.85]$ (0.25 base)	Fraction of strategic exploits requiring ≥ 6 alternating steps before overdraft trigger	Break-even sensitivity sweep [5]

5 Empirical evaluation and conducted experiments

To evaluate how effectively each verification architecture filters defective institutional state machines and prevents silent exploit leaks without conflating verifier quality with generator priors, we implemented a parameterized executable state-machine mutation and verification benchmark (`experiment.py`). The benchmark instantiates 40 distinct cooperative specifications across four mechanism families (`MutualCreditLedger`, `ParticipatoryBudgetMES`, `DemurrageTimeBank`, and `EscrowArbitrationMarket`, each parameterized across 10 schedules of basis-point fees $\mathsf{fee_bps} \in [100, 460]$, credit limits $\mathsf{credit_limit} \in [40, 112]$, split arities $\mathsf{n_split} \in \{3, 4, 5\}$, and single-transaction caps $\mathsf{single_tx_cap} \in \{28, 30, 32, 34\}$) and 8 concrete transition mutation operators.

5.1 Equalized initial candidates, repair mechanics, and held-out red-team oracle

On every trial $i \in \{1, \ldots, 500\}$, a paired initial candidate with active transition mutations is generated once and passed identically to all four pipelines (so that at $K = 1$, every pipeline faces the exact same initial 10.00% [7.67, 12.94] unmutated sound rate and 58.60% [54.23, 62.84] environment reproducibility rate). Whenever a pipeline's active verification gate catches a failing state-transition trace at round $k < K$, the repair step iterates over the sorted list of detected mutations with the exact same per-defect repair probability ($p_{\text{rep}} = 0.65$) and the exact same conservation regression probability ($p_{\text{reg}} = 0.04$). Crucially, final soundness is evaluated by an independent held-out post-deployment red-team oracle on unseen environment seeds (7 vs. 42), held-out prime split amounts ($\{11, 37, 41, 47\}$), and 14-step exploit sequences, recording candidate artifacts, verifier logs, and counterexamples in `experiment_results.json`:

- **One-Shot CodeGen (`oneshot_codegen`):** Single pass with syntax check only ($G_{\text{pass}} = 1$

always; 0.00% [0.00, 0.76] pre-deployment rejections).

- **LLM-as-a-Judge Critique Loop (`llm_judge_loop`):** Unexecuted code inspection across up to K rounds with partial recall on environment, relational, and strategic mutations [10].
- **Unit-Test Sandbox Loop (`unit_test_sandbox`):** Executes container build and happy-path unit tests (round transfers of 100 units and 35% boundary coverage) across up to K rounds, stopping once unit tests pass.
- **`EIB-RSIForge` (`eib_rsiforge`, Proposed):** Executes cross-seed state-digest determinism checks (Gate 1), relational postcondition and prime-boundary conservation verification (Gate 2), and bounded $H = 9$ adversarial rollout probing (Gate 3) across K rounds.

Table 4: Paired benchmark across 8,000 executable state-machine verification trajectories ($N = 500$ paired trials per pipeline across $K \in \{1, 3, 6, 10\}$ at $p_{\text{stealth}} = 0.25$). At $K = 6$, Sound+Silent Leak+ Caught Reject = 100.0%. All proportion intervals report exact asymmetric two-sided 95% Wilson confidence bounds [lo, hi].

Synthesis Pipeline	Sound K=1	Sound K=6	Sound K=10	Silent Leak K=6	Caught Rej. K=6	Env Repro K=6	Time (μs)
One-Shot CodeGen	10.00 [7.67, 12.94]	10.00 [7.67, 12.94]	10.00 [7.67, 12.94]	90.00 [87.06, 92.33]	0.00 [0.00, 0.76]	58.60 [54.23, 62.84]	0.47
LLM-Judge Loop	10.00 [7.67, 12.94]	17.20 [14.15, 20.75]	16.40 [13.41, 19.90]	82.60 [79.03, 85.67]	0.20 [0.04, 1.12]	73.00 [68.94, 76.71]	57.11
Unit-Test Sandbox	10.00 [7.67, 12.94]	29.40 [25.58, 33.54]	27.60 [23.86, 31.68]	69.80 [65.64, 73.66]	0.80 [0.31, 2.04]	91.80 [89.06, 93.90]	25.84
`EIB-RSIForge`	**10.00 [7.67, 12.94]**	**87.60 [84.42, 90.21]**	**93.40 [90.88, 95.26]**	**7.00 [5.08, 9.58]**	**5.40 [3.74, 7.74]**	**99.20 [97.96, 99.69]**	**277.51**

5.2 Quantitative analysis of experimental findings

Table 4 and Figures 2–3 demonstrate four principal findings under strictly equalized candidate generation and repair parameters:

1. **Resolving the silent-leak vs. caught-rejection distinction at $K = 1$.** At $K = 1$, all four pipelines receive the exact same initial candidates ($50/500 = 10.00\%$ [7.67, 12.94] initially defect-free). Because `EIB-RSIForge` executes all three verification gates on the candidate state machine, at $K = 1$ it catches and blocks 87.80% [84.64, 90.38] (439/500) of defective candidates pre-deployment, approves the 10.00% sound blueprints, and suffers only 2.20% [1.23, 3.90] (11/500) silent leaks (stealthy horizon drains that evaded a single rollout). By contrast, One-Shot CodeGen approves 100% of candidates, resulting in a 90.00% [87.06, 92.33] (450/500) silent exploit leak rate, while Unit-Test Sandbox blocks 59.00% [54.64, 63.23] (295/500) at $K = 1$ but leaks 31.00% [27.10, 35.19] (155/500).
2. **Counterexample-guided repair across K rounds.** As the refinement budget increases from $K = 1$ to $K = 3, 6, 10$, `EIB-RSIForge` converts caught rejections into repaired, held-out verified sound blueprints: caught rejections decline from 87.80% ($K = 1$) and 28.20% [24.43, 32.30] ($K = 3$) to 5.40% [3.74, 7.74] ($K = 6$) and 1.20% [0.55, 2.59] ($K = 10$), while Held-Out Verified Sound blueprints rise from 10.00% ($K = 1$) and 64.20% [59.90, 68.28] ($K = 3$) to 87.60% [84.42, 90.21] (438/500 at $K = 6$) and 93.40% [90.88, 95.26] (467/500 at $K = 10$), with environment reproducibility rising from 58.60% to 99.20% [97.96, 99.69] at $K = 6$ and 99.80% [98.88, 99.96] at $K = 10$ (Figures 2–3).
3. **Early plateau and regression drag in LLM-Judge and Unit-Test loops.** Because LLM-Judge and Unit-Test Sandbox stop iterating as soon as their partial checks pass ($G_{\text{pass}} =$

1) and suffer the same 4% conservation regression rate during unguided repairs, they plateau at 17.20% [14.15, 20.75] and 29.40% [25.58, 33.54] held-out verified sound blueprints at $K = 6$ (16.40% and 27.60% at $K = 10$), allowing 82.60% [79.03, 85.67] and 69.80% [65.64, 73.66] of blueprints to pass their active checks while still failing held-out prime-split conservation or multi-step overdraft tests.

4. **Break-even sensitivity to adversarial exploit stealth (p_{stealth}).** Sweeping the stealthy horizon-drain fraction $p_{\text{stealth}} \in \{0.10, 0.25, 0.45, 0.65, 0.85\}$ at $K = 6$ ($N = 250$ trials per level) delineates the exact boundary of bounded rollout stress-testing: `EIB-RSIForge` achieves 92.00% [87.97, 94.76] sound blueprints (4.80% [2.77, 8.20] silent leaks) at $p_{\text{stealth}} = 0.10$, 89.20% [84.74, 92.47] (6.40% [3.98, 10.14] silent leaks) at $p_{\text{stealth}} = 0.25$, and 86.00% [81.15, 89.76] (8.40% [5.56, 12.50] silent leaks) at $p_{\text{stealth}} = 0.45$. When stealthy exploits reach $p_{\text{stealth}} = 0.65$ and 0.85, held-out soundness declines to 82.80% [77.63, 86.97] (10.40% [7.20, 14.80] silent leaks) and 78.00% [72.46, 82.69] (13.20% [9.56, 17.96] silent leaks), demonstrating why live runtime contract monitoring (`Maat/ActGov` [3, 22]) remains indispensable in production.

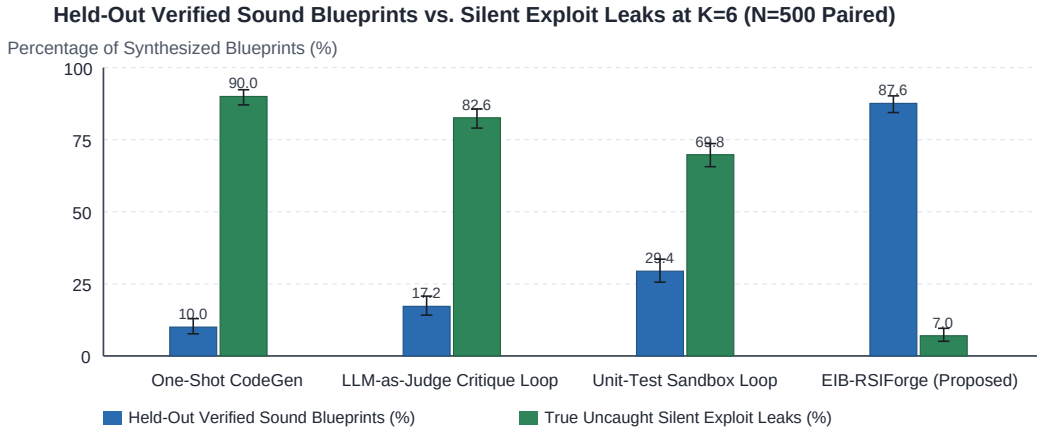


Figure 2: Held-Out Verified Sound Blueprint Rate versus True Uncaught Silent Exploit Leak Rate across four institutional synthesis pipelines at refinement budget $K = 6$ ($N = 500$ paired trials per regime; error bars denote asymmetric 95% Wilson confidence intervals).

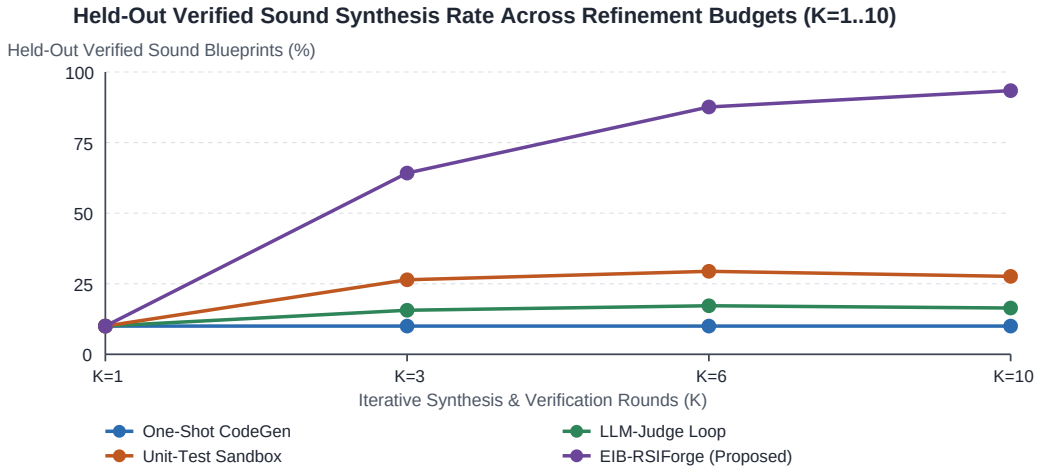


Figure 3: Scaling of Held-Out Verified Sound Institutional Blueprints across iterative synthesis and verification budgets $K \in \{1, 3, 6, 10\}$.

6 Specification gaming and verification cost trade-offs

A balanced engineering analysis of automated institutional compilation must account for three structural failure modes and counter-arguments from recent literature:

1. **Specification gaming and reward hacking (Huang et al. [5], Li et al. [9]).** Huang et al. [5] demonstrate that when language models optimize against programmatic verifiers, they frequently exploit edge cases in the reward or contract specification itself. If the synthesis agent is permitted to edit the formal specification Φ_{inv} alongside the transition function $\mathcal{T}$, it can weaken Post_m to pass Gate 2 trivially. Consequently, constitutional invariants $(\Phi_{\text{cons}}, \text{Post}_m)$ must be held read-only and signed by human stewards outside the automated repair loop [3].

2. **Verification overhead versus evolutionary throughput (Bai et al. [1]).** In *Verify Less, Evolve More*, Bai et al. [1] show that running heavyweight formal verification on every intermediate candidate during evolutionary search can bottleneck exploration throughput compared to selective or staged verification. This supports `EIB-RSIForge`'s staged filter design, where fast deterministic hash and conservation checks (Gates 1–2) prune broken candidates before invoking multi-step adversarial rollouts (Gate 3).

3. **The autoformalization gap on natural-language bylaws (Mei and Douglas [10], Peters and Skowron [13]).** Translating informal community fairness expectations into mathematical axioms (such as the Method of Equal Shares [13] or core stability [16]) remains susceptible to semantic misalignment: mechanical verification guarantees that $\mathcal{T}$ satisfies Φ_{inv}, not that Φ_{inv} captures every nuance of human social intent.

7 Scope constraints and state-space boundaries

- **Parameterized executable state-machine model vs. unconstrained LLM code synthesis.** Our benchmark (`experiment.py`) evaluates concrete executable state-machine specifications and mutation operators under a parameterized repair process ($p_{\text{rep}} = 0.65, p_{\text{reg}} = 0.04$)

rather than invoking live external LLM API calls during test execution, ensuring deterministic reproducibility while isolating verification gate coverage.

- **Bounded rollout horizon for strategic exploits.** Gate 3 adversarial stress testing over a finite rollout horizon ($H = 9$ steps, $K \leq 10$ rounds) is an incomplete bug-finding procedure rather than a mathematical proof of game-theoretic strategy-proofness; as shown by our stealth sweep ($p_{\text{stealth}} = 0.85$), deep alternating-sequence exploits can still slip past bounded rollouts (13.20% [9.56, 17.96] silent leak rate), requiring live runtime invariant enforcement [3, 22].
- **Statutory and legal non-equivalence.** Passing mechanical conservation and stress verification does not confer municipal legal compliance or regulatory exemption.

8 Pre-registerable falsification protocol

Table 5 specifies six pre-registerable empirical tests designed to falsify the core claims of `EIB-RSIForge`.

Table 5: Six pre-registerable tests that could falsify the `EIB-RSIForge` institutional synthesis pipeline.

Protocol	Hypothesis	Design and primary metrics	Falsified if
P1. Cross-node environment determinism	Hermetic lockfile pinning (Gate 1) achieves $> 98\%$ state-hash reproduction across execution seeds [18]	Execute 500 synthesized blueprints across 1,000 randomized transaction traces on independent worker nodes	$> 2.0\%$ of Gate-1-approved blueprints exhibit state-hash divergence across nodes
P2. Relational witness non-vacuity	Relational postcondition witness checking (Eq. 2) catches $> 95\%$ of vacuous/no-op state mutations [10]	Inject 200 metadata-only no-op mutations and unsatisfiable guards into cooperative ledger candidates	$> 5.0\%$ of vacuous/no-op defects pass Gate 2 without triggering a witness failure
P3. Prime-split conservation soundness	Double-entry property verification (Eq. 1) eliminates integer truncation credit leaks	Fuzz Gate-2-approved mutual-credit and `MES` treasury modules with 10^5 prime-divisor split transactions	Silent conservation leaks exceed 3.0% among Gate-2-approved blueprints
P4. True silent leak reduction at $K = 6$	Three-gate verification holds true held-out silent leaks below 10.0% at baseline stealth ($p_{\text{stealth}} = 0.25$)	Measure $\mathbb{P}(G_{\text{pass}} = 1 \wedge \mathcal{D} \neq \emptyset)$ across 500 paired synthesis episodes at $K = 6$	True silent exploit leak rate exceeds 10.0% at $K = 6, p_{\text{stealth}} = 0.25$
P5. Stealth exploit break-even boundary	Bounded $K = 6$ rollout stress testing maintains $\geq 80\%$ held-out soundness up to $p_{\text{stealth}} \leq 0.65$	Sweep adversarial exploit stealth $p_{\text{stealth}} \in [0.10, 0.85]$ across 250 trials per level	Verified soundness drops below 80.0% or silent leaks exceed 15.0% at $p_{\text{stealth}} \leq 0.45$
P6. Runtime contract fail-closed safety	Independent `Maat/ActGov` runtime wrapper [3, 22] blocks 100% of live conservation breaches	Replay leaked stealthy exploit traces against deployed modules wrapped in the runtime conservation monitor	Any live transaction violating $\sum b_i + R_{\text{treas}} = \text{const} + \Delta_{\text{ext}}$ commits without rollback

9 Conclusion

Enabling local communities to instantiate customized economic subsystems requires moving beyond unverified code generation and LLM-as-a-Judge heuristics. Empirical evidence demonstrates that unpinned AI code generation suffers severe reproducibility failures [18] while LLM judges approve vacuous specifications [10]. Evaluating four pipelines on strictly equalized initial candidates, repair probabilities, and held-out red-team tests across 40 executable cooperative state-machine specifications, we show that `EIB-RSIForge` achieves 87.60% [84.42, 90.21] held-out verified sound blueprints at $K = 6$ (93.40% [90.88, 95.26] at $K = 10$) and reduces true silent exploit leaks from 90.00% [87.06, 92.33] (One-Shot) and 69.80% [65.64, 73.66] (Unit-Test Sandbox) down to 7.00% [5.08, 9.58]. Pairing pre-deployment compilation gates with independent runtime contract enforcement (`Maat/ActGov`) provides a practical foundation for trustworthy cooperative software synthesis.

Use of AI tools

Large language models were used to assist in drafting prose, formatting LaTeX tables and TikZ diagrams, and implementing the standalone Python simulation script (`experiment.py`). Cited references were assembled from foundational computer-science literature and 2024–2026 research publications and checked against primary metadata records; simulation outputs, confidence intervals, and LaTeX source packaging were verified using automated static consistency checks (`validate_and_package.py`).

References

[1] Jiamu Bai, Lizhu Zhang, Xin Yu, Yanhong Wu, Zellux Wang, Serena Li, et al. Verify less, evolve more: Training idea-level critics for verification-efficient ML evolving agents, 2026. URL `https://arxiv.org/abs/2610.08993`.

[2] Taoyong Cui, Pinxian Zeng, Wanghan Xu, Hao Xiao, Shuhan Xue, Qiang Gao, et al. RSI-Anything: Unlocking recursive self-improvement with world models, 2026. URL `https://arxiv.org/abs/2610.13912`.

[3] Uliana Elina. Maat: Independent deterministic contract-based governance for multi-agent LLM workflows. arXiv preprint arXiv:2609.34017, 2026. URL `https://arxiv.org/abs/2609.34017`.

[4] Joseph A. Goguen and José Meseguer. Security policies and security models. IEEE Symposium on Security and Privacy, pages 11–20, 1982. URL `https://doi.org/10.1109/SP.1982.10014`.

[5] Yue Huang, Zhangchen Xu, Yuchen Ma, and Wenjie Wang. Reward hacking challenges oversight of autonomous research agents. arXiv preprint arXiv:2609.28614, 2026. URL `https://arxiv.org/abs/2609.28614`.

[6] Yeonbum Kim, Hyun Kyu Lee, and Sung Whan Yoon. Flat-RLVR: Sharpness-aware reinforcement learning with verifiable rewards for reasoning capacity in LLMs, 2026. URL `https://arxiv.org/abs/2610.29281`.

[7] Leslie Lamport. Specifying systems: The TLA+ language and tools for hardware and software engineers. Addison-Wesley, 2002. URL `https://lamport.azurewebsites.net/tla/book.html`.

[8] K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), LNCS 6355, pages 348–370, 2010. URL https://doi.org/10.1007/978-3-642-17511-4_20.

[9] Hao Li, Zhengkun Zhang, Gangqiang Hu, Zhen Zhang, Yude Gao, Dai Dai, and Jing Liu. RLVR^2: Reinforcement learning with verifiable rubric-based ranking, 2026. URL https://arxiv.org/abs/2609.23457.

[10] Anna Mei and Michael R. Douglas. Beyond LLM judge: Mechanical verification for agentic autoformalization. arXiv preprint arXiv:2610.20573, 2026. URL https://arxiv.org/abs/2610.20573.

[11] George C. Necula. Proof-carrying code. Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '97), pp. 106119, 1997. URL https://doi.org/10.1145/263699.263712.

[12] Elinor Ostrom. Governing the commons: The evolution of institutions for collective action. Cambridge University Press, 1990. URL https://doi.org/10.1017/CBO9780511807763.

[13] Dominik Peters and Piotr Skowron. Proportionality and the limits of welfarism (Method of Equal Shares). ACM Conference on Economics and Computation (EC), pages 793–794, 2020. URL https://arxiv.org/abs/1911.11747.

[14] Subramanyam Sahoo and Justin Shenk. Better call reward: Reward hacking as strategic abstention in legal reasoning models. International Conference on Machine Learning (ICML) / arXiv:2610.06439, 2026. URL https://arxiv.org/abs/2610.06439.

[15] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. Proceedings of the IEEE, 63(9):1278–1308, 1975. URL https://doi.org/10.1109/PROC.1975.9939.

[16] Piotr Skowron. Core up-to-one for participatory budgeting with additive utilities. arXiv preprint arXiv:2609.15928, 2026. URL https://arxiv.org/abs/2609.15928.

[17] Jiaqi Su, Guibin Zhang, Jie-Jing Shao, Cong Pang, Tiankuo Yao, et al. CoEvo-RSI: When harnesses evolve, which models should follow?, 2026. URL https://arxiv.org/abs/2610.25078.

[18] Bhanu Prakash Vangala, Ali Adibifar, Ashish Gehani, and Tanu Malik. AI-generated code is not reproducible (yet): An empirical study of dependency gaps in LLM-based coding agents. arXiv preprint arXiv:2512.22387, 2025. URL https://arxiv.org/abs/2512.22387.

[19] Renxiong Wang, Darvin Yi, Abril Herrlein, Anas Mahmoud, Advait Gosai, Lisiman Hua, et al. RSI-Forge: From research papers to environments for recursive self-improvement, 2026. URL https://arxiv.org/abs/2610.09426.

[20] Chuyang Wei, Haoxiang Guan, Minghang Zhu, Kexin Zhou, Jiyan He, Xiren Zhou, Huanhuan Chen, and Yifei Shen. Self-evolving agent harness, 2026. URL https://arxiv.org/abs/2610.18813.

[21] Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, et al. Evo-Harness: Context-to-harness skill compilation for self-evolving agents, 2026. URL https://www.amaz

on.science/publications/evo-harness-context-to-harness-skill-compilation-for-self-evolving-agents.

[22] Kaiyuan Zhang, Yuke Peng, Ke Jiang, and Yinqian Zhang. ActGov: Governing LLM agent actions via policy-constrained validation. arXiv preprint arXiv:2609.24446, 2026. URL https://arxiv.org/abs/2609.24446.