HARVEST: HIERARCHICALLY ADAPTIVE RTL VERIFICATION VIA STRUCTURED EPISODE TRAJECTORIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Automatic testbench generation with large language models can reduce the cost of hardware verification, but a testbench that compiles and passes its own checks may still miss critical bugs. This creates a fundamental challenge for automated verification: the signal used to decide whether a generated testbench is adequate cannot simply be the agent's own report of success. We introduce HARVEST, a framework for adaptive testbench generation that routes designs among heterogeneous verification agents and uses an external adequacy signal, mutation kill rate, to identify insufficient testbenches and trigger complementary verification. A contextual bandit selects among three agents targeting structural, semantic, and adversarial coverage. When the selected testbench has low external adequacy, HARVEST escalates to a second agent and composes the resulting testbenches into a single suite, preserving complementary tests rather than discarding the first attempt. On 100 open-source Verilog designs, we find that 13% of testbenches that pass their own checks nevertheless kill fewer than one-third of injected bugs, revealing a substantial gap between agent-reported success and actual verification adequacy. Routing to a single agent achieves 91.7% of the oracle quality, while adequacy-gated escalation further improves quality at modest additional cost. Upon acceptance we will release 300 structured trajectories spanning the 100 designs as a benchmark for adaptive verification under external adequacy signals. Together, these results show that external adequacy can provide a reliable control signal for detecting false-confidence testbenches and coordinating complementary verification agents.

1 INTRODUCTION

Large language model (LLM) agents can now generate RTL testbenches automatically, which is attractive because writing testbenches by hand is one of the largest costs in hardware verification. We study this task (automatic testbench generation) and the reliability problem it exposes. A generated testbench can compile and pass its own assertions yet still miss the majority of real bugs, and the agent's reported success does not reveal this. Mutation testing provides an adequacy signal independent of the agent's claims: a testbench with low mutation kill rate (MKR) is inadequate regardless of its pass rate. In our corpus, 13% of passing testbenches (30 of 237) have MKR < 0.3, kill fewer than one in three injected RTL bugs, and achieve only 56.5% mean line coverage despite reporting no failures (false confidence). Existing work (Orenes-Vera et al., 2023; Kande et al., 2023; Meng et al., 2026) offers no mechanism to detect this failure mode at runtime.

When agent success signals are unreliable, two questions arise: (a) can we detect low-adequacy outcomes at runtime without a ground-truth oracle? and (b) when the first agent's output is inadequate, can we route to a more effective alternative? RTL verification is an ideal testbed for both: Verilog has formal semantics, mutation injection is a well-established independent adequacy measure (Vemuri et al., 1997), and three qualitatively distinct generation strategies arise naturally from hardware design structure (Section 3.3): a structural agent (FSM/coverage-driven) that drives state-transition coverage; a semantic agent (contract/specification-driven) that checks functional correctness against the spec or inferred port contracts; and an adversarial agent (boundary/corner-case-driven) that stresses all-zeros, all-ones, max-count, and competing control signals. Across 100 open-source Verilog designs these strategies exhibit strong per-design variance (48% of designs have a best-vs-worst reward gap exceeding 0.20; Section 5.2), and, critically, static routing performs worse than

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

always using the best fixed strategy: strategy effectiveness is latent in execution outcomes, not source-level features. We identify a key empirical link: low MKR on the first strategy predicts high strategy-selection uncertainty ($r = -0.363$, $p < 0.001$; robust to a coverage-only reward that excludes MKR, Section 5.3). This inadequacy signal flags a poor outcome and indicates that escalation is worth its cost.

We introduce HARVEST (**H**ierarchically **A**daptive **R**TL **V**erification via **S**tructured **E**pisode **T**rajectories). HARVEST does not try to build a better single testbench generator; it treats each agent's generation quality as given and instead allocates test-time compute across a portfolio of the three generation agents above, routing and escalating. It formalizes this as a test-time compute allocation problem: given a portfolio of heterogeneous agents and an external adequacy signal, learn to route designs to the most effective agent and escalate when adequacy is low. Routing sends each design to the agent most likely to produce an adequate testbench, learned from execution outcomes because static source features do not predict the winner. MKR-gating detects at runtime when the routed testbench is inadequate and escalates to a second, qualitatively different agent, then composes both testbenches into one suite. A LINUCB contextual bandit, the Bandit Router (Figure 1), learns routing from structured trajectories, and an MKR-gated Meta Controller invokes the Bandit Router's second-ranked strategy when MKR $< \tau = 0.5$, treating low adequacy as the trigger. These are the two tiers that make HARVEST hierarchically adaptive: the Bandit Router picks which agent to run first, and the Meta Controller sits above it, observes the outcome, and decides whether to escalate and compose (a control-decision hierarchy, distinct from the RTL submodule hierarchy of a design). It is adaptive because this second-tier decision allocates test-time compute per design at runtime, conditioned on the observed adequacy signal: high-MKR designs stop after one run, and only the low-MKR designs earn a second agent. This external-signal gating is the key distinction from self-confidence routing: a confident-but-wrong model never escalates in a cascade, whereas in HARVEST a testbench that passes its own assertions but kills few mutations escalates regardless of agent confidence, and the second run is a qualitatively different bandit-selected strategy, not an i.i.d. resample. The formulation is intentionally simple to isolate the value of the adequacy signal itself.

Our key contributions are as follows: (1) We formulate automated testbench generation as a sequential allocation problem over heterogeneous verification agents and introduce MKR, an external adequacy signal for identifying false-confidence testbenches missed by coverage-based or self-confidence-based gating. MKR enables adaptive routing through a meta-controller, achieving the effectiveness of joint strategy execution while avoiding unnecessary computation. (2) We show that adequacy-guided composition of independently generated testbenches provides statistically significant iso-cost improvements over selecting the strongest individual strategy, demonstrating that effective routing requires external validation beyond internal confidence signals. (3) We introduce a benchmark of 300 structured agent trajectories across 100 RTL designs, enabling systematic evaluation of routing, escalation, and decision-making under external adequacy feedback.

2 RELATED WORK

LLM-based and agentic HW verification. LLMs can generate syntactically valid testbenches (Thakur et al., 2024; Liu et al., 2023; Kande et al., 2023), and multi-turn agents that iterate on simulation/coverage feedback improve them (Orenes-Vera et al., 2023); VeriGen (Thakur et al., 2024) narrows the gap to frontier models on generation. HAVEN (Meng et al., 2026) extends this to UVM testbenches on an AES core; our plain-Verilog adversarial strategy reaches 99.75% line coverage on the same IP, within 0.25% of the 100% oracle testbench (Section 6). We use RealBench's (Jin et al., 2025) expert testbenches (100% line coverage) as oracle upper bounds. All this work optimizes a single generation attempt; HARVEST instead treats generation quality as fixed and poses the strategy-routing problem across three qualitatively distinct agents. Our adversarial arm draws on the coverage-directed test-generation tradition in hardware fuzzing (Laeufer et al., 2018; Canakci et al., 2021) but with LLM-generated stimuli. MKR as a testbench-quality metric is well-established (Vemuri et al., 1997) and has been used for LLM testbench filtering (Kande et al., 2023); our distinct contribution is to use it online as a runtime signal for verification uncertainty (low MKR marks designs whose optimal strategy is unpredictable from static features), transforming a passive score into an active routing trigger.

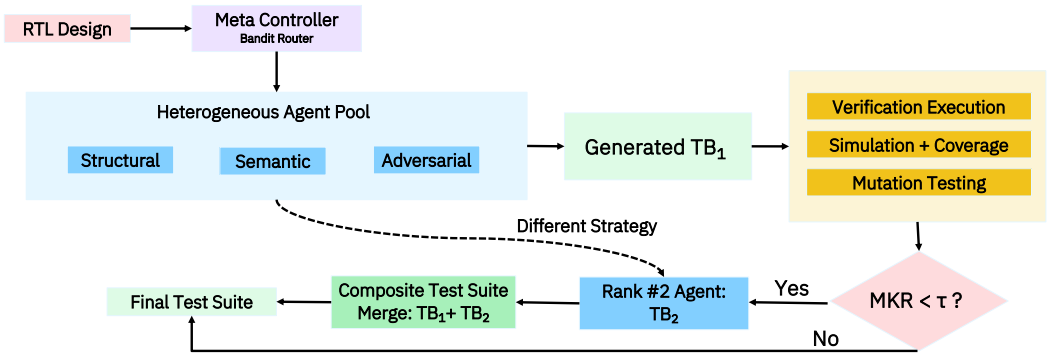


Figure 1: HARVEST architecture. A control plane routes each design to one of three heterogeneous agents via a LinUCB bandit, then gates on the run's mutation kill rate (MKR): the reliable 66% keep the single run; the low-adequacy 34% (MKR $< \tau = 0.5$) run a second agent and compose the two testbenches rather than select one.

Algorithm selection, bandits, and LLM routing. Strategy routing is an instance of algorithm selection (Rice, 1976), applied to solvers by SATzilla (Xu et al., 2008) and AutoFolio (Lindauer et al., 2015) and to hyperparameter optimization by SMAC3 (Lindauer et al., 2022); HARVEST differs in that its arms are autonomous agents varying qualitatively (not parametrically) and its routing is augmented by a runtime execution signal (MKR) absent from classical selection. We learn routing with LINUCB (Li et al., 2010), a standard contextual bandit suited to our low-feature regime. LLMs have been proposed as zero-shot algorithm selectors (Tornede et al., 2023); our LLM-designated baseline instantiates this and is beaten by the execution-grounded bandit (improvement CI $[+0.003, +0.132]$), confirming that LLMs encode useful priors but cannot replace models that observe execution outcomes. Most related is LLM routing and inference-time compute allocation (Chen et al., 2024; Ong et al., 2025; Snell et al., 2025): FrugalGPT cascades models by cost until a quality threshold is met, RouteLLM routes on preference-trained proxies, and the best-of-N paradigm samples repeatedly and keeps the best (Brown et al., 2024). These route across homogeneous generators using the agent's self-confidence; HARVEST routes across heterogeneous agents and escalates on an external adequacy signal (MKR), so a confident-but-wrong testbench that kills few mutations still escalates (Section 1).

3 THE HARVEST FRAMEWORK

3.1 PROBLEM FORMULATION

Figure 1 gives the overall pipeline: a control plane routes each design to one of three agents and, when the run's adequacy signal is low, escalates and composes a suite. We now formalize each component. Let $\mathcal{D}$ be a corpus of RTL designs, each consisting of one or more Verilog source files and an optional natural-language specification. For each design $d \in \mathcal{D}$, we have a finite set of strategies $\mathcal{S} = \{\text{structural}, \text{semantic}, \text{adversarial}\}$. Running strategy s on design d produces an episode $\tau = (s, d, \mathbf{x}, r, \text{trace})$ where $\mathbf{x} \in \mathbb{R}^6$ is a context vector derived from d prior to any agent execution, $r \in [0, 1]$ is the episode reward, and trace is the multi-turn agent transcript. The strategy selection problem is: given a new design d' with context $\mathbf{x}'$, select $s^* = \arg\max_{s \in \mathcal{S}} \mathbb{E}[r \mid s, \mathbf{x}']$.

3.2 CONTEXT VECTOR

The context vector $\mathbf{x} \in \mathbb{R}^6$ is computed from static analysis of the RTL source before any agent runs, in under 100 ms via regex with no EDA tool dependency. Its features are `has_fsm` (clocked FSM present; 66% of designs; strongest ablation signal, Δregret $+0.032$ when dropped), `has_protocol` (6 designs), `seq_fraction`, `branch_density`, `complexity_norm`, and `rtl_lines_norm` (feature definitions in Appendix A). These replaced three degenerate v1 features (all constant or near-constant on the corpus), lowering bandit LOO regret $0.131 \rightarrow 0.122$ (Section 7). Crucially,

static features do not determine which strategy wins: the routing signal is latent in execution outcomes (coverage, MKR), not source structure, motivating both the Bandit Router and the MKR-gated Meta Controller (Section 5.3).

3.3 Agent Strategies

Each strategy is implemented as a multi-turn LLM agent with access to two EDA tools: `iverilog_sim` (Verilator-based simulation) and `yosys_mutate` (Yosys-based structural analysis and mutation generation). All three strategies follow a common **Plan** $\rightarrow$ **Generate** $\rightarrow$ **Verify** $\rightarrow$ **Reflect** loop: the agent first plans its test intent from the RTL structure or specification, generates a testbench, verifies it via simulation and coverage measurement, then reflects on gaps and refines, up to two iterations within a 20-minute wall-clock budget.

Strategy-specific lenses. The structural agent applies this loop through a coverage-driven lens: it extracts the state transition graph via Yosys and plans sequences to drive every reachable FSM transition, prioritizing uncovered branches in each reflection step. The semantic agent plans from the module specification or inferred port contracts, generating tests that verify functional correctness for each specified use case and iterating on contract violations. The adversarial agent targets corner cases and boundary conditions (all-zeros, all-ones, maximum count values, simultaneous competing control signals) informed by coverage-directed test generation (Laeufer et al., 2018).

MKR as execution-time guidance. After each strategy run, the agent reports a mutation kill rate (MKR) computed by a Yosys-based mutation pass on a seed set of injected RTL bugs. MKR serves two roles: as a component of the reward signal (Equation 1, weight 0.10), and as the trigger for the MKR-gated Meta Controller (Section 5.3), which runs the Bandit Router's second-ranked strategy when the first achieves MKR $< \tau = 0.5$. Appendix H walks through how the three lenses diverge on one concrete design (`aes_cipher_top`), where the boundary-driven adversarial strategy reaches 99.75% line coverage (the oracle-best arm) while the FSM- and contract-driven strategies target different code regions.

3.4 Reward Function

The episode reward $r \in [0, 1]$ is a weighted combination of four sub-scores:

$$r = 0.40 \cdot r_{\text{pass}} + 0.35 \cdot r_{\text{line}} + 0.15 \cdot r_{\text{toggle}} + 0.10 \cdot r_{\text{mkr}} \tag{1}$$

where r_{pass} is the testbench simulation pass rate, r_{line} and r_{toggle} are line and toggle coverage normalized to $[0, 1]$, and r_{mkr} is the mutation kill rate on the reward-seed mutations. To verify that MKR findings are not circular, we also define a coverage-only reward $r_{\text{no-mkr}}$ that excludes the MKR term (renormalized to $[0, 1]$); all spread-ratio analyses in Section 5.3 are replicated on it and yield equivalent conclusions. A held-out collusion probe guards against agents gaming MKR by reading the mutation seeds; zero episodes were flagged in the current corpus (derivation and collusion mechanism in Appendix C).

3.5 LinUCB Bandit

We learn the context-to-strategy mapping with the disjoint LinUCB contextual bandit (Li et al., 2010) (one arm per strategy, $d = 6$, $\alpha = 0.5$), using the standard per-arm $A_a \leftarrow A_a + \mathbf{x}\mathbf{x}^\top$, $b_a \leftarrow b_a + r\mathbf{x}$ updates and UCB selection. LinUCB is chosen for the small, interpretable feature space and its flat α sweep (Section 7), not for novelty: the key claim is the gating principle, not LinUCB. Crucially, we fit it **offline via batch replay** over all scored (context, strategy, reward) triples. In this batch regime the UCB exploration term α is inert (the alpha sweep is exactly flat), so LinUCB reduces to ridge regression; the online regret bounds do not apply and we claim no regret guarantee. All reported regret values are empirical leave-one-out estimates, not theoretical bounds.

4 Experimental Setup

Our evaluation is organized around four questions. **RQ1 (Overall performance):** does learned, execution-grounded routing plus adequacy-gated escalation reduce strategy-selection regret relative to LLM-inferred routing (and, directionally, hand-coded and fixed-best baselines)? (Section 5.2.)

RQ2 (Adequacy signal and escalation): is MKR a usable runtime quality and escalation signal, is the escalation gain non-circular, and does composing the two run testbenches beat selecting one? (Section 5.3.) **RQ3 (What drives routing):** which context features carry the routing signal, and how sensitive is the result to the model, feature set, training-set size, and exploration parameter? (Feature importance in Section 5.4; ablations in Section 7.) **RQ4 (Generalization):** do the strategies and the controller transfer to a harder external corpus of real-world IP designs? (Section 6.)

Design Corpus

The primary corpus consists of 100 open-source Verilog designs collected from OpenCores and related GitHub repositories, spanning simple peripherals, GPIO controllers, CRC engines, arithmetic units, and protocol bridges (29–484 RTL lines). The corpus yields 300 analyzed episodes (100 designs $\times$ 3 strategies, 294 of which carry a valid score), evaluated under leave-one-out cross-validation. All trajectories were generated with Claude Sonnet 4.6 ; each strategy was run independently, with no agent observing another strategy's testbench or coverage at generation time. To assess generalization beyond the OpenCores corpus, we additionally evaluate on four RealBench designs (Jin et al., 2025): two from the AES cryptographic family and two from the SD card controller subsystem. RealBench provides expert-written testbenches achieving 100% line coverage for each design, which serve as oracle upper bounds. All three HARVEST strategies are run on each design; results are reported as case studies in Section 6. The primary corpus skews toward simple peripherals: `has_golden_vec` is zero for all 100 designs, and `has_protocol` is positive for only 6 designs (6%), limiting generalization claims. The RealBench case studies begin to address this gap. An illustrative sample spanning the three oracle-best strategies and the difficulty range, together with the full 100-design manifest, appears in Appendix A (Tables 5 and 6).

Baselines

We compare against five baselines. The **oracle** always selects the highest-reward strategy per design and represents the upper bound. The **LLM-designated rule** uses a zero-shot frontier LLM prompt to map static RTL features (FSM presence, sequential fraction, protocol presence) to strategy choices (protocol $\rightarrow$ adversarial; FSM/high-seq $\rightarrow$ structural; otherwise $\rightarrow$ semantic), representing the best achievable static routing without execution data. The **heuristic rules** apply the same mapping via a hand-coded decision tree, representing domain expert knowledge. **Fixed-best** always selects semantic (the highest single-arm corpus mean, Table 1; regret in Table 2), representing the strongest single-arm prior. Finally, **round-robin** cycles through all three strategies uniformly, representing the no-routing-signal baseline.

Metrics

Our primary metric is mean LOO regret: $\frac{1}{N}\sum_{i=1}^{N}(r_{\text{oracle},i} - r_{\text{method},i})$, where lower is better, with bootstrap 95% confidence intervals from $B = 5000$ samples. We also report the zero-regret rate (fraction of designs where the selected strategy matches oracle), mean reward of the selected strategy, and mean line coverage and MKR. Wall-clock cost in seconds per design is reported in Table 2 for methods with deterministic cost. All 100 designs have full three-arm observations, making the leave-one-out procedure equivalent to offline policy evaluation on a full-information dataset: the oracle is directly observable and regret estimates need no off-policy correction. Online deployment would require a separate exploration phase and incur additional regret before context vectors accumulate, so the LOO results are an upper bound on deployable performance at $n = 100$ designs. Claims are scoped to these 100 open-source Verilog designs; generalization to industrial or larger RTL corpora requires additional evaluation.

5 RESULTS

5.1 TRAJECTORY COLLECTION STATISTICS

Table 1 summarizes the episodes collected across 100 Verilog designs. We analyze 300 episodes (100 designs $\times$ 3 strategies), of which 237 pass simulation. Not all 300 carry a valid score: 6 of the 300 design/strategy pairs produced no scored trajectory (a compile or simulation error, or no valid score), leaving 294 scored episodes, exactly 98 per strategy. This is why per-strategy counts are 98 and not 100. The SIM-PASS column reports the fraction of the 294 scored episodes that pass simulation, so its "All" value (82.7%) is taken over 294 rather than the 300 analyzed.

Table 1: Trajectory corpus statistics across 100 Verilog designs. Rows cover the scored episodes retained per strategy (98 each, 294 total; of 300 analyzed, 237 pass simulation); "All" aggregates the three rows. SIM-PASS is the fraction of scored episodes that pass simulation (over 294, not 300). Mean reward is the composite score from Equation equation 1.

Strategy	Episodes	Sim-pass	Mean line cov	Mean MKR	Mean reward	Mean cost (s)
Structural	98	81.0%	67.7%	56.2%	0.712	857
Semantic	98	85.7%	64.5%	60.5%	0.716	857
Adversarial	98	81.6%	58.5%	54.2%	0.666	905
All	294	82.7%	63.9%	56.9%	0.699	873

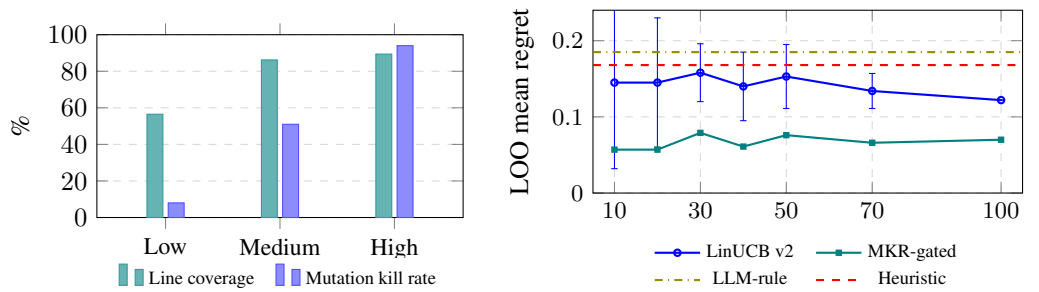
Table 2: Strategy selection under leave-one-out cross-validation ($n = 100$); regret is oracle minus selected reward (lower is better), 95% CIs bootstrap ($B = 5000$). Lower block: iso-cost escalation baselines. MKR-gated survives on the coverage-only reward $r_{\text{no-mkr}}$ (regret 0.073), so the result is not circular. † mean seconds per design; n/a marks deterministic selection with no agent cost.

Method	Mean regret ↓	95% CI	Mean reward	Cost† (s)
Oracle (upper bound)	0.000	n/a	0.841	2619
MKR-gated (ours)	**0.070**	[0.038, 0.108]	0.771	∼1160
Contextual bandit (route only)	0.122	[0.079, 0.170]	0.719	873
Fixed-best (semantic)	0.139	[0.091, 0.189]	0.702	857
LLM-designated rule	0.185	[0.122, 0.247]	0.656	n/a
Heuristic rules	0.168	[0.109, 0.235]	0.673	n/a
Round-robin mean	0.173	[0.140, 0.211]	0.668	873
Escalation baselines: each gate runs the bandit's second-ranked arm on the flagged designs (matched to 34% where noted) and keeps the better run; composite reward.				
Always-top-2 (all designs)	0.059	[0.028, 0.097]	0.782	∼1746
MKR-gated (ours)	0.070	[0.038, 0.108]	0.771	∼1160
Coverage-gated (lc $< 70\%$)	0.068	[0.037, 0.106]	0.773	∼1115
Pass-fail gated (fail on 1st)	0.082	[0.049, 0.122]	0.759	∼1100
Random escalation (34%)	0.116	[0.073, 0.164]	0.725	∼1160
Bandit (1-run)	0.122	[0.079, 0.170]	0.719	873

Strategy variance: per-design stakes are high. For the same design, the three strategies produce substantially different rewards: 48% of designs have reward spread > 0.20 (mean 0.361, SD 0.343). Average reward is similar across strategies (Friedman $p = 0.214$), but per-design best-vs-worst differs strongly (Wilcoxon $p < 0.001$): the actionable variance is in which strategy wins each design, not between strategies in aggregate, motivating a context-adaptive selector. Pairwise effect sizes are small (Cohen's $d \leq 0.12$; full distributions in Appendix B). Structural, semantic, and adversarial are oracle-best for 39%, 38%, and 23% of designs, respectively; no strategy dominates, motivating learned selection.

5.2 STRATEGY SELECTION RESULTS (RQ1)

Routing to a single strategy cuts compute by 67%, and adequacy-gated escalation then reaches 91.7% of oracle reward at $\sim$1.33$\times$ single-run compute (§5.3); the routing decision itself, at $n = 100$, ties the simplest baselines. Table 2 reports LOO regret with bootstrap 95% CIs and Wilcoxon p-values against oracle. The LINUCB bandit achieves mean regret **0.122**, significantly beating the LLM-generated rule (CI $[+0.003, +0.132]$), but its gain over the hand-coded heuristic and fixed-best-semantic is only directional (CI $[-0.008, +0.133]$, $p = 0.57$). The LLM-designated rule, using static features optimally under a zero-shot prompt, performs worse than fixed-best: only by observing execution outcomes can a learner identify which features predict the reward differential.



(a) Quality by MKR tier ($n = 30$ / 42 / 165 episodes).

(b) LOO regret vs. training-set size (20 subsamples/size, SD bars).

Figure 2: (a) Testbench quality rises with MKR tier (237 passing episodes): low-MKR runs pass simulation yet reach only 56.5% line coverage (false confidence). (b) Escalate-and-compose (MKR-gated) beats routing at every training-set size and stays near 0.07, while LinUCB routing improves only weakly ($0.145 \rightarrow 0.122$); the advantage comes from merging a second strategy, not from better routing (static rules drawn at full-corpus regret).

5.3 MKR as a Runtime Verification Uncertainty Signal (RQ2)

The MKR-gated Meta Controller is the second tier over the Bandit Router: it observes the routed run's outcome and adaptively allocates a second agent only where warranted, which is why the two-tier policy wins where a flat one cannot. The external MKR signal's distinct, irreplaceable value is trustworthiness: it flags false-confidence testbenches (high coverage, low kill rate) that a coverage or self-confidence gate is blind to by construction, and escalating on it also lowers regret relative to the self-confidence pass/fail analogue that prior routing and cascade work relies on (detailed below). The efficiency payoff follows from targeted escalation: while flat learned routing only ties the best fixed strategy (0.122 vs. 0.139 regret, not significant), spending a second agent on the $\sim 1/3$ of designs the gate flags cuts regret to 0.070, a statistically significant improvement over the one-run bandit (95% CI $[+0.023, +0.086]$, $p = 0.0003$) and a large reduction over the best fixed strategy, at a $\sim 1.33\times$ average compute overhead versus $3\times$ for always running all three. Targeted allocation, not selection alone, delivers this regret number. The controller is motivated by a key empirical discovery: mutation kill rate (MKR) is a reliable runtime predictor of verification uncertainty, the degree to which the optimal strategy depends on execution outcomes rather than static structural features.

MKR predicts reward spread: Across all 100 designs, per-design mean MKR correlates negatively with reward spread (Pearson $r = -0.363$, $p < 0.001$): designs whose testbenches catch few mutations have high cross-strategy variance, while high-MKR designs are strategy-agnostic. The relationship is monotone in the threshold (Appendix D, Table 10): at $\tau = 0.5$ low-MKR designs have $2.1\times$ higher reward spread (0.507 vs. 0.246; Mann-Whitney $p = 0.002$), rising to $5.6\times$ at $\tau = 0.7$; the ratio replicates at $2.2\times$ ($p = 0.002$) on the coverage-only reward $r_{\text{no-mkr}}$, ruling out circularity.

MKR as a testbench trustworthiness signal: MKR is also a direct quality indicator for the testbench itself: Figure 2a stratifies passing episodes into a monotone quality gradient by MKR tier ($r = 0.381$ between MKR and line coverage), so a single runtime measurement tells the engineer whether to trust the outcome without any ground-truth testbench.

MKR-gated Meta Controller: The two-stage policy runs the bandit's first pick, then, if MKR $< \tau = 0.5$, runs the second-ranked strategy and keeps the better result, triggering on 34/100 designs. [1] Regret for this and the iso-cost baselines is in Table 2. Escalating on pass/fail (the self-confidence analogue used by prior cascade work) reaches 0.082 versus 0.070 for MKR-gating. MKR's value is structural: it flags a false-confidence mode pass/fail cannot represent by construction, though at $n = 100$ this subset is small (13% of designs), so MKR- and coverage-gating are statistically indistinguishable as triggers here; an empirical advantage over coverage-gating would need a corpus enriched for false-confidence designs, left to future work. We scope the gate as a cost mechanism:

[1] The 34 here counts designs whose routed-arm MKR falls below 0.5 (the gate trigger); the 44 in the spread analysis counts designs whose per-design mean MKR falls below 0.5, different quantities over the same corpus.

always-top-2 reaches 0.059 but runs two agents on every design ($\sim$1746 vs. $\sim$1160 compute), trading $\sim$0.011 regret for $\sim$34% compute savings. This is not a circularity artifact: on $r_{\text{no-mkr}}$ (MKR excluded), regret is essentially unchanged (0.073).

Escalation as suite composition: compose rather than discard: The baselines above all select the better of two runs, but the strategies are complementary specialists (the line-best and MKR-best arms differ on 40/100 designs), so once escalation has run both, we compose them: merging the testbenches post-hoc rather than discarding one. This merge beats keep-max by a small but significant margin (+0.0008, 95% CI $[+0.0001, +0.0018]$) at zero added compute, a free iso-cost win. A per-dimension-max union-coverage floor (Appendix F) further shows an oracle combined suite reaches regret 0.011, a $\sim 6\times$ cut from 0.070, indicating substantial headroom in how the second testbench could be generated.

5.4 Feature Importance and Theta Weights (RQ3)

The learned $\hat{\theta}_a$ (Table 3) show `rtl_lines_norm` dominating all arms (0.82–1.14), consistent with larger designs being harder to cover. `has_fsm` carries the most discriminative signal ($+0.202$ structural, -0.021 adversarial), confirmed by the feature ablation (Δregret $+0.032$ when dropped). The sequential features (`seq_fraction`, `branch_density`) also carry meaningful weight absent from the v1 degenerate features.

Table 3: LinUCB $\hat{\theta}_a$ weights per arm (enriched v2 features, fit on all 100 designs): `rtl_lines_norm` dominates; `has_fsm` and `seq_fraction` carry the discriminative signal.

Arm	fsm	prot	seq	brn	cx	lines
Structural	$+0.20$	$+0.09$	$+0.16$	$+0.09$	$+0.04$	$+0.82$
Semantic	$+0.23$	-0.06	-0.01	-0.12	$+0.15$	$+1.07$
Adversarial	-0.02	$+0.01$	$+0.18$	$+0.05$	-0.01	$+1.14$

Coverage and MKR by Design Complexity: Smaller designs consistently achieve higher coverage and MKR (Table 9), given fewer code paths per iteration budget. Oracle-best splits 39/38/23 across structural, semantic, and adversarial, yet no static feature predicts the winner (Appendix G.1). Routing significantly beats fixed-best only on the low-complexity tier (95% CI $[+0.003, +0.117]$), ties it elsewhere, where all strategies struggle together, motivating MKR-gated escalation.

6 RealBench Case Studies (RQ4)

RealBench (Jin et al., 2025) is an RTL code-generation benchmark whose expert-written testbenches reach 100% line coverage per IP core. We repurpose four of its IP designs (AES core family and SD card controller subsystem, all absent from the primary corpus) as a harder testbench-generation evaluation, asking whether our agents approach oracle quality on designs more complex than the OpenCores corpus. On `aes_cipher_top`, the adversarial strategy reaches 99.75% line coverage, within 0.25% of the oracle. On `aes_sbox` (combinational), structural reaches 100% line and toggle coverage. On both SD designs, structural reaches 100% line coverage with MKR 0.85 and 1.00, which the controller correctly treats as high-confidence single-run cases. These confirm Harvest generalizes beyond OpenCores, including to cryptographic datapaths. `aes_sbox` also shows the limit of a single adequacy signal: MKR is undefined for its purely combinational logic (no sequential state to mutate), so the controller falls back to coverage rather than distrusting a good run.

7 Ablation Studies (RQ3)

We summarize the ablation conclusions here; full tables, the learning curve, and the per-setting bandit comparison are in Appendix G. **Feature dropout** (Table 13): `has_fsm` carries the strongest discriminative signal (Δregret $+0.032$ when dropped), followed by `rtl_lines_norm` ($+0.008$) and `complexity_norm` ($+0.006$); dropping the near-degenerate `has_protocol` reduces regret by -0.009. The enriched v2 features lower LOO regret $0.131 \rightarrow 0.122$ over the v1 dead-feature set.

Table 4: HARVEST best-strategy results on four RealBench IP cores (pass rate 100% for all). ‡`aes_sbox` is purely combinational, so mutation testing injects no flip-flop mutants and MKR is undefined rather than 0; the run is kept on its saturated coverage. §HAVEN's ≈91% is a composite coverage figure, not directly comparable to our line coverage; shown for context only.

Design	RTL lines	Line cov	Toggle cov	MKR	Best strategy
`aes_cipher_top`	541	**99.75%**	98.82%	1.00	adversarial
`aes_sbox`	266	**100.0%**	100.0%	n/a‡	structural
`sd_clock_divider`	45	**100.0%**	100.0%	0.85	structural
`sd_crc_7`	36	**100.0%**	98.41%	1.00	structural
HAVEN (Meng et al., 2026) (AES, UVM; composite cov)		≈91%§	n/a	n/a	n/a
Oracle (ground-truth testbench)		100%	100%	n/a	n/a

Bandit comparison (Appendix E): MKR-gating reaches 0.070 ($p = 0.0003$ vs. the one-run bandit) at $\sim$1.33$\times$ compute; the per-method regrets and CIs are in Table 2. **Learning curve** (Figure 2b): averaged over 20 random subsamples per training size, routing regret improves only weakly and noisily with data (0.145 at n=10 to 0.122 at n=100; per-subsample SD falls from 0.11 to 0.02), consistent with the corpus's limited feature diversity. Escalation-and-compose stays near 0.07 at every training size, well below all single-run routing baselines. **Strategy subset** (Table 14): every 2-strategy subset has higher regret than the full three arms; semantic+adversarial (0.132) is near-optimal while structural is the least marginal arm at this scale. **Alpha sensitivity**: LOO regret is identical (0.122, zero-regret 43/100) across $\alpha \in \{0.1, 0.25, 0.5, 1.0, 2.0\}$, confirming α is inert in the batch-fit regime where feature vectors are nearly identical across designs.

8 Discussion

Connection to algorithm selection. HARVEST's core mechanism (instance features feeding a portfolio selector) is algorithm selection (Xu et al., 2008; Lindauer et al., 2015), here applied to LLM-based RTL verification routing. Unlike solver portfolios with rich instance features (Xu et al., 2008), small-tier RTL features collapse to essentially two discriminative dimensions (§G.2), characterizing what feature richness routing needs rather than a failure of the approach. Because learned routing ties fixed-best here, our stronger positive result is about composition, not selection. **Limitations:** Results are scoped to the small-complexity tier of open-source Verilog at $n = 100$ designs, so fine-grained significance claims (e.g., bandit vs. fixed-best) should be read as directional pending a larger, more diverse corpus; our RealBench case studies (§6) provide preliminary evidence of generalization beyond this tier. MKR is undefined for purely combinational logic with no sequential mutants, a known coverage-domain restriction for which the controller falls back to coverage rather than failing.

9 Conclusion

We introduced HARVEST, casting automated RTL verification as a test-time compute allocation problem over heterogeneous agents. Across 100 Verilog designs, three LLM agent strategies show substantial per-design variance, though learned routing ties fixed-best at this scale. MKR detects false-confidence outcomes and correlates with strategy-selection uncertainty; the real win is MKR-gated escalation (regret 0.070), on top of which composing the two run testbenches, rather than keeping the better one, gives a small but significant free iso-cost gain. A per-dimension-max union floor suggests an oracle ceiling of $\sim$6$\times$, but selection provably cannot reach it, and the true per-mutant union equals the floor on all measurable designs, so realizing it needs conditional generation, not selection. We plan to release, upon acceptance, the 300 structured trajectories to accelerate follow-on work.

Reproducibility Statement

The HARVEST pipeline, trajectory collection scripts, bandit training code, and evaluation scripts are included as supplementary material. The trajectory dataset (300 episodes across 100 designs,

schema_version = 2) will be provided upon acceptance. The context-vector extraction code (six v2 features derived from the RTL source via fast regex plus port-name analysis; Section 3) is fully deterministic given the same RTL source. The LINUCB hyperparameter $\alpha = 0.5$ was selected on the 5-design pilot validation set (in-sample; the LOO results in Table 2 are insensitive to α). All designs in the corpus are publicly available from OpenCores and cited GitHub repositories.

AI USE STATEMENT

In this work, we used Claude Sonnet 4.6 as theverification agents that generate testbenches and interact with EDA tools. The agents are the subject of the study, not assistants in writing the paper. All experimental results were generated by the automated pipeline without manual selection or cherry-picking of episodes. We take responsibility for all claims and results in this work.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. It involves no human subjects, no personal or sensitive data: the design corpus consists entirely of publicly available open-source RTL from OpenCores and cited GitHub repositories, used in accordance with their licenses. The method generates verification testbenches for hardware designs; we see no direct avenue for harmful misuse beyond that of general hardware-verification tooling. The dataset we plan to release contains only generated testbenches, coverage metrics, and mutation-testing statistics for these public designs, with no proprietary or identifying content.

REFERENCES

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. In *arXiv preprint arXiv:2407.21787*, 2024.

Sadullah Canakci, Leila Delshadtehrani, Furkan Eris, Michael Bedford Taylor, Manuel Egele, and Ajay Joshi. DirectFuzz: Automated test generation for RTL designs using directed coverage-based fuzzing. In *Proceedings of the 58th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2021.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research (TMLR)*, 2024.

Pengwei Jin et al. RealBench: Benchmarking Verilog generation models with real-world IP designs. *arXiv preprint arXiv:2507.16200*, 2025.

Rahul Kande, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, and Jeyavijayan Rajendran. LLM-assisted generation of hardware assertions. In *arXiv preprint arXiv:2306.14027*, 2023.

Kevin Laeufer, Jack Koenig, Donggyu Kim, Jonathan Bachrach, and Koushik Sen. RFUZZ: Coverage-directed fuzz testing of RTL on FPGAs. In *Proceedings of the 37th IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2018.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pp. 661–670, 2010.

Marius Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. AutoFolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Marben, Tim Müller, Philipp Schlede, Stefan Falkner, and Frank Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

Yun-Da Liu, Cheng-Yi Wu, Pin-Yu Ho, and Chung-Yang Huang. RTLFixer: Automatically fixing RTL syntax errors with large language models. In *arXiv preprint arXiv:2311.16543*, 2023.

Chang-Chih Meng, Yu-Ren Lu, Guan-Yu Lin, Tsung Tai Yeh, et al. HAVEN: Hybrid automated verification ENgine for UVM testbench synthesis with LLMs. *arXiv preprint arXiv:2604.27643*, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

Marcelo Orenes-Vera, Neel Manocha, David Wentzlaff, and Margaret Martonosi. From RTL to SVA: LLM-assisted RTL design verification. In *arXiv preprint arXiv:2309.09437*, 2023.

John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally improves reasoning. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. VeriGen: A large language model for Verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3), 2024.

Tanja Tornede, Alexander Tornede, Jonas Hanselle, Felix Möller, Marcel Wever, and Eyke Hüllermeier. AutoML in the age of large language models: Current challenges, future opportunities and risks. In *arXiv preprint arXiv:2306.08107*, 2023.

Ranga Vemuri, Rajesh Mohan, and Sagar Thummalapalli. Mutation-based testbench evaluation for HDL designs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(4):433–443, 1997.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

A Design Corpus Details

The six context features (Section 3) are defined as follows: `has_fsm` (clocked FSM present), `has_protocol` (port names match an AHB/AXI/Wishbone/I2C/SPI/UART/PCIe vocabulary), `seq_fraction` (sequential/total `always` blocks), `branch_density` (`case`/`if` per code line, a cyclomatic proxy), `complexity_norm` (log-scaled FF+mux count), and `rtl_lines_norm` (log-normalized line count). Table 6 lists all 100 Verilog designs in the corpus, their detected features, and oracle-best strategy. All designs are single-file Verilog (hence `file_count_norm` is constant at 0 across all 100 corpus designs), and `has_golden_vec` (column G) is zero for every design. The complete design manifest with repository URLs is provided in `train_manifest.jsonl` in the dataset to be released upon acceptance.

B Additional Result Tables

Table 7 gives the per-strategy reward distributions and pairwise variance analysis summarized in Section 5.2; Table 8 reports the per-metric verification-quality breakdown referenced in Section 5.3; Table 9 reports the design-size breakdown referenced in the results.

C Reward Function Details

The composite reward (Equation equation 1) combines four sub-scores. r_{pass} is the fraction of test vectors that pass, and r_{line}, r_{toggle} are line and toggle coverage normalized to $[0, 1]$. Lint pass/fail and the collusion-detection probe run in parallel but do not directly enter the reward; they gate episode acceptance and flag anomalies respectively.

Table 5: Illustrative sample of the design corpus (rows drawn verbatim from the full 100-design table, Table 6), spanning all three oracle-best strategies, the FSM/protocol feature range, and including a timeout/failure case. Features: F=has_fsm, P=has_protocol, G=has_golden_vec (0 for all designs). Reward 0.00 indicates timeout or simulation failure across all three strategies.

Design	F	P	G	Oracle	Best reward
BRSFmnCE	1	0	0	semantic	1.000
PWM	1	0	0	semantic	0.963
Nx1_mux	0	0	0	structural	1.000
present_encryptor_top	1	0	0	structural	1.000
AHB2WB	1	1	0	adversarial	0.941
cfg_crc	1	0	0	adversarial	0.996
rc4	1	0	0	adversarial	0.994
ima_adpcm_dec	1	1	0	structural	0.997
eqspiflash	1	0	0	adversarial	0.768
deflate	0	0	0	structural	0.000

Coverage-only reward for circularity checks. To verify that findings involving MKR are not circular (MKR contributes 0.10 to r), we define a coverage-only reward excluding the MKR term, renormalized so the remaining weights sum to one:

$$r_{\text{no-mkr}} = \frac{0.40 \cdot r_{\text{pass}} + 0.35 \cdot r_{\text{line}} + 0.15 \cdot r_{\text{toggle}}}{0.90} \tag{2}$$

All spread-ratio analyses in Section 5.3 are replicated on $r_{\text{no-mkr}}$ and yield equivalent conclusions (ratio $2.2\times$, $p = 0.002$ at $\tau = 0.5$).

Collusion detection. To prevent agents from gaming the mutation kill rate by reading the mutation seeds, we hold out a probe set of 20 single-line mutations (operators: STUCK0, STUCK1, FLIP, SKIP) drawn with a disjoint random seed. Episodes where the agent-reported Mkr exceeds 0.70 on this held-out set are flagged; the mutation sub-reward is zeroed and the episode is marked for inspection. Of all collected episodes, zero were flagged by the collusion detector. The mechanism is validated by design and does not alter any reward in the current corpus.

D MKR Threshold Analysis

Table 10 gives the full MKR threshold sweep referenced in Section 5.3: reward spread (best-vs-worst strategy gap per design) for designs below and above each threshold, with Mann-Whitney p-values and the coverage-only replication.

E Full Bandit Algorithm Comparison

Table 11 gives the full method comparison across all evaluation settings, including the v1 (dead-feature) bandit variant. It reproduces the regret ordering of the main-text regret table (Table 2). Notably, fixed-best (semantic) matches the oracle on more designs than the bandit (zero-regret 46/100 vs. 43/100) yet has higher mean regret (0.139 vs. 0.122), because it loses badly on the designs it misses.

F Escalation as Suite Composition

Section 5.3 reframes escalation from selection (keep the better of two runs) to composition (merge the two runs into a suite). This appendix gives the full analysis behind that reframing. All numbers use the composite reward, $\alpha = 0.5$, bootstrap $B = 5000$ (seed 42), and are budget-identical to keep-max escalation (the second run is already paid for).

Precondition: the arms cover different faults. Composition can only help if the strategies are genuinely complementary. They are: in 40/100 designs the line-coverage-best arm is a different arm than the MKR-best arm, and the mean within-design spread across the coverage dimensions (line/toggle/branch/MKR) is ~ 0.45. Merging two testbenches is therefore not the same operation as picking the better one.

Combined-suite reward is a conservative floor. We define the combined-suite reward as the per-dimension maximum of the two run arms: a merged testbench covers a line/toggle if either component does, and its union kill rate is at least the max of the two component kill rates. This is a conservative floor on the true merged suite, not an exact union: the exact union kill rate requires per-mutant kill vectors, which the frozen corpus does not store (only the scalar `mutation_kill_rate`; `per_file_results` is empty). We therefore re-ran the mutation harness to recover per-mutant kill vectors and measure the true union directly (next paragraph); on the six designs where it is measurable the true union equals the floor exactly, so on that subset the floor is empirically exact rather than a lower bound. We also stress that the trajectories were generated independently: each strategy ran blind to the others, so this is composition of independently-generated testbenches (an ensemble / mixture-of-experts sense of "collaboration"), not interactive collaboration during generation.

Three findings. (1) The oracle ceiling is large but is an upper bound, not a lever. The oracle combined suite cuts regret from 0.070 (keep-max) to 0.011 ($\sim 6\times$) and beats the best single arm on 6/100 designs (negative regret). The same story replicates on the coverage-only reward $r_{\text{no-mkr}}$: keep-max 0.073 $\rightarrow$ oracle-combine 0.015. This $\sim 6\times$ is an oracle per-dimension-max across all four coverage dimensions; the measurement below shows the MKR portion of it is not realizable. (2) A free iso-cost win. Because escalation already runs both testbenches, merging them instead of discarding the loser gives a small but statistically significant gain over keep-max (+0.0008, 95% CI $[+0.0001, +0.0018]$, excludes 0) at zero extra cost. Modest in magnitude, but free and directionally consistent. (3) Selection is not the lever (honest limitation). Which complementary arm to add is essentially unlearnable from static RTL context at this scale: the complementary-best second arm equals the scalar-second-best arm 86% of the time (19/22 triggered designs), and choosing the complementary one rather than the scalar-second gains only ~ 0.0007 on average. A learned complementarity predictor (6 features, $n = 100$) does not beat keep-max (0.078, CI overlaps).

Measuring the true union kill rate (a negative result). The floor above upper-bounds the MKR dimension by the per-arm max. To test whether the true per-mutant union exceeds it, we re-ran the mutation harness on the trigger designs to recover per-mutant kill vectors and computed the exact union of the two arms' kill sets. Of the 35 trigger designs reproduced from our pipeline (the frozen paper reports 34/100; the small difference is version/seed drift in re-running the pipeline (Verilator/Yosys versions plus the n=100 vs. n=104 corpus drift), not a reproducibility bug), only 6 are validly measurable: 12 error out (VHDL-only repos, a missing testbench for the chosen arm, or no mutation candidates) and 17 have a failing unmutated baseline on at least one arm, where a kill rate would be an artifact. On all six measurable designs the true per-mutant union kill rate equals the per-dimension-max floor exactly (gap $+0.0000$): the two arms' kill sets are nested (one is a subset of the other), so there is zero per-mutant complementarity in fault detection on this subset. A functional-only kill variant (excluding compile-break kills) gives the same $+0.0000$ gap. Substituting the true union for the floor over all designs leaves the merge gain at rounding scale (its 95% CI includes 0). The MKR portion of the $\sim 6\times$ ceiling is therefore not realizable: the floor is empirically exact on the measurable subset, not a lower bound. Caveats a reviewer should weigh: $n = 6$ valid measurements is thin (29 of 35 excluded for the reasons above); several valid designs have small mutant sets (as few as 2–7 usable mutants); and some sit near MKR saturation or have a strongly dominant arm, so the nested structure of this subset may not represent the 29 excluded designs. We report this as a genuine negative result for the MKR dimension; our mutation-union measurement and its per-design outputs will be released with the data-preparation artifacts upon acceptance.

Toward the oracle: future work. Closing the gap to the 0.011 oracle needs information the current corpus does not provide (per-mutant kill vectors to compute exact union coverage) and, more fundamentally, a generation procedure that makes the second run complementary by construction: generating the second testbench conditioned on the first strategy's coverage gaps, rather than running it blind. Selection among three independently-generated arms provably cannot reach the oracle,

because the complementary arm coincides with the scalar-second-best arm 86% of the time; the remaining headroom lives in how the second testbench is produced. We flag conditional/interactive generation as the most promising route and leave it to future work.

G Ablation Details

This appendix gives the full ablation tables and learning curve summarized in Section 7.

Feature dropout. Table 13 shows LOO regret when each context feature is zeroed out in turn (enriched v2 vector). Dropping `has_fsm` costs $+0.032$ regret, the strongest individual signal; `rtl_lines_norm` costs $+0.008$ and the sequential features (`seq_fraction` $+0.001$, `complexity_norm` $+0.006$) carry signal absent from the v1 dead features. Dropping `has_protocol` reduces regret by -0.009: active for only 6/100 designs, it adds slight noise in the disjoint model at this scale.

Training set size (learning curve). The learning curve (Figure 2b, main text) is computed by drawing 20 random subsamples of each training size, running full LOO within each subsample, and averaging the mean regret. Routing regret is noisy and only weakly monotone in data: $0.145 \rightarrow 0.145 \rightarrow 0.158 \rightarrow 0.140 \rightarrow 0.153 \rightarrow 0.134 \rightarrow 0.122$ at $n = 10, 20, 30, 40, 50, 70, 100$, with per-subsample SD shrinking from 0.11 (n=10) to 0.02 (n=70). The bandit sits below the LLM rule (0.185) at every size but is within noise of the hand-coded heuristic (0.168) until $n \geq 70$. Escalation-and-compose (MKR-gated) is far lower throughout (0.057–0.079, mean ≈ 0.07), and its advantage does not depend on training-set size, consistent with the finding that the gain comes from running and merging a second strategy, not from better learned routing.

Strategy subset. Table 14 compares all three 2-strategy subsets against the full system. All pairs have higher regret than the three-arm bandit; semantic+adversarial (0.132) nearly matches all three (0.122), while dropping adversarial hurts more (structural+semantic 0.156, structural+adversarial 0.141), so structural is the least marginal arm at this corpus scale (its `has_fsm` signal is partially captured by `rtl_lines_norm`).

G.1 Corpus Feature Distribution

The current corpus exhibits severe feature degeneracy that prevents meaningful strategy differentiation. `has_golden_vec` is zero for all 100 designs; the semantic agent's primary routing signal is entirely absent from the corpus. `has_protocol` is positive for only 6 of 100 designs (6%); the adversarial agent's primary routing signal contributes minimal discrimination. Only `has_fsm` (positive for 66/100 designs, 66%) and `rtl_lines_norm` (continuous, varying) carry discriminative information. This degeneracy explains the flat learning curve (Figure 2b) and the near-parallel theta vectors (Table 3): with two effective dimensions and three strategies, the bandit cannot learn strategy-specific routing.

A corpus including designs with protocol interfaces (AXI/Wishbone bridges, AHB peripherals), designs with golden simulation vectors, and designs with rich FSM structure (RISC-V cores, memory controllers) would provide the feature diversity needed for learned routing to outperform heuristics reliably.

G.2 When Does Each Strategy Win?

Oracle-best assignments split 39/38/23 across structural, semantic, and adversarial, but no static feature reliably predicts the winner within this near-uniform corpus. Structural wins most on FSM designs, yet semantic's principal routing signal (golden availability) is masked because `has_golden_vec` is zero for all 100 designs, and adversarial's 6 protocol designs show no consistent dominance at this scale. This is the corpus feature-degeneracy that underlies the flat learning curve and near-parallel theta vectors (§5.4): with only two effective dimensions (`has_fsm`, `rtl_lines_norm`) across three strategies the bandit cannot learn strategy-specific routing (Appendix G.1). Routing's advantage is not uniform, though: stratified by complexity, learned routing significantly beats fixed-best on the low-complexity tier (95% CI $[+0.003, +0.117]$, excludes 0),

where a learnable per-design signal exists, while on the medium- and high-complexity tiers it merely ties fixed-best. On the harder designs the three strategies struggle together and no selection policy wins, which is precisely what motivates the MKR-gated escalation.

Six designs (6%) received zero reward across all three strategies, all from exceeding the agent timeout rather than incorrect generation; a handful more scored low and failed simulation on every strategy. These are addressable with a larger iteration budget (per-design rewards in Appendix A).

H Qualitative Strategy Differences: A Worked Example

To illustrate how the three strategies differ in practice, we walk through a single concrete design, `aes_cipher_top` (541 lines, an AES-128 core with a key scheduling FSM and hierarchical submodules), one of the four RealBench case studies (Section 6, Table 4). All three strategies are run on this design for the walkthrough; because the oracle-best run saturates MKR at 1.00 (well above $\tau = 0.5$), the MKR-gated controller treats it as a high-confidence single-run case and does not escalate.

The three strategies attack the same RTL through qualitatively different lenses. The **structural** strategy is FSM-transition-driven: it extracts the state-transition graph via Yosys and plans stimulus sequences that drive every reachable transition of the key-scheduling FSM, prioritizing uncovered branches on each reflection. The **semantic** strategy is specification/contract-driven: it plans from the AES round-function contract (plaintext/key in, ciphertext out after the specified number of rounds), generating tests that check functional correctness of the cipher rather than exhaustively walking control state. The **adversarial** strategy is boundary-driven: it targets corner cases of the datapath (all-zero and all-one keys and plaintexts, maximal round counters, simultaneous control-signal contention), informed by coverage-directed test generation.

On this design the boundary-driven adversarial strategy is oracle-best, reaching **99.75%** line coverage, 98.82% toggle coverage, and MKR 1.00 (values from Table 4), within 0.25% of the 100% oracle testbench on the same IP (HAVEN reports ≈91% composite coverage, a different axis, not directly comparable). Intuitively, the AES datapath's hardest-to-cover behavior is at value boundaries in the round transformation rather than in FSM control flow, so the boundary lens exposes coverage the FSM- and contract-oriented lenses reach less directly; the high, saturated MKR is why the gate treats one strong run as sufficient rather than escalating further.

Sourcing note. The per-strategy line/toggle/MKR breakdown for the structural and semantic arms on `aes_cipher_top` is not reported as a per-design row in the paper's tables (only the winning adversarial arm's coverage is tabulated), so the relative behavior of the two non-winning arms is described qualitatively from the strategy definitions (Section 3.3) rather than with specific coverage numbers. Only the adversarial arm's coverage (99.75% / 98.82% / MKR 1.00) and the oracle-best assignment are sourced directly from Table 4; we do not assign numeric coverage to the structural or semantic runs to avoid fabricating figures the corpus does not record.

I LinUCB Alpha Sensitivity

As reported in Section 7, the LOO regret is identical across all tested α values: mean regret = 0.122 and zero-regret rate = 43/100 at $\alpha \in \{0.1, 0.25, 0.5, 1.0, 2.0\}$. This confirms that α is inert in the batch-fit regime: the UCB exploration bonus does not meaningfully reorder arms when feature vectors are nearly identical across designs (dominated by `rtl_lines_norm`).

J Agent Prompt Template (Structural Strategy)

Below is an abbreviated version of the system prompt used for the structural agent strategy. The semantic and adversarial prompts follow the same structure with strategy-specific instructions replacing the structural guidance.

```
You are a hardware verification expert.
Your task: generate a Verilog testbench for the module
described below using a STRUCTURAL strategy.
```

```
STRUCTURAL STRATEGY:
1. Use Yosys (yosys_mutate tool) to analyze the RTL and
   identify all FSM states and transitions.
2. Generate test vectors that exercise every reachable
   FSM state and every branch in the RTL.
3. Use iverilog_sim to compile and simulate. Measure
   line coverage. Iterate to cover uncovered branches.
4. Generate mutations with yosys_mutate and verify the
   testbench kills each mutation.

At the end of your session, print exactly:
HARVEST_RESULT: pass=N fail=M total=T
  line_cov=X.X toggle_cov=Y.Y branch_cov=Z.Z
  mutation_kill=K.KK
(mutation_kill is a FRACTION 0.0-1.0, not a percentage)
```

Table 6: Full design corpus (100 Verilog designs, small tier). Features: F=has_fsm, P=has_protocol, G=has_golden_vec. Oracle: best strategy by composite reward. Reward 0.00 indicates timeout or simulation failure across all three strategies. Entries that repeat a name (e.g. `i2c_gpio`, `spigpio`, `top`) are distinct designs from distinct repositories that happen to share a top-module name; all 100 come from distinct sources and none is double-counted.

Design	F	P	G	Oracle	Best reward
ACC	1	0	0	semantic	0.994
AHB2WB	1	1	0	adversarial	0.941
AIC1106_PCM	1	1	0	adversarial	0.911
AntiLog2	1	0	0	semantic	0.995
BRSFmnCE	1	0	0	semantic	1.000
DPSFnmCE	1	0	0	semantic	0.992
FIR_cascaded	0	0	0	structural	0.916
HashChain	0	0	0	structural	0.845
Log2flowthru	1	0	0	semantic	0.975
Nx1_mux	0	0	0	structural	1.000
PREFIX	0	0	0	semantic	0.953
PREFIX_regfile	1	0	0	semantic	1.000
PWM	1	0	0	semantic	0.963
SSPx_Slv	1	0	0	semantic	0.993
StepperMotorDrive	0	0	0	semantic	0.405
TB_SPI_MasSlv	0	0	0	structural	0.976
a429_rx_iface	1	0	0	semantic	0.952
aes_core	0	0	0	semantic	0.963
agc	1	0	0	semantic	0.877
alttop	1	0	0	structural	0.862
auto_oscillator	1	0	0	semantic	0.975
bcd_to_binary	1	0	0	semantic	0.955
bitscan	0	0	0	semantic	1.000
bublesort	1	0	0	structural	0.949
carry_generate	0	0	0	structural	1.000
cfg_crc	1	0	0	adversarial	0.996
clk_div	1	0	0	adversarial	0.986
clk_valid	1	0	0	structural	1.000
copyright	0	0	0	adversarial	1.000
cordic	0	0	0	semantic	0.869
cr_div_tb	1	0	0	structural	0.942
crc32d16N	0	0	0	adversarial	0.973
d_TopLevel	1	0	0	semantic	0.827
debouncer	1	0	0	structural	1.000
deflate	0	0	0	structural	0.000
divider	0	0	0	structural	0.997
dpll	0	0	0	adversarial	0.899
dpram	0	0	0	adversarial	1.000
dragonball_wbmaster	1	0	0	structural	0.000
eqspiflash	1	0	0	adversarial	0.768
fht	1	0	0	structural	0.938
gpio	1	0	0	semantic	0.946
i2c	0	0	0	structural	0.836
i2c_gpio	1	0	0	structural	0.908
i2c_gpio	1	0	0	structural	0.930
i2crepeater	1	0	0	adversarial	0.953
ibufdff	0	0	0	semantic	1.000
ima_adpcm_dec	1	1	0	structural	0.997
interrupt_controller	1	0	0	semantic	0.973
ioport_a16_d8_wo	1	0	0	structural	0.973
lcd_16x2_8bit	1	0	0	semantic	0.969
lcd_wishbone_slave	0	0	0	adversarial	0.942
logsig_approx	0	0	0	semantic	0.500
mod3	1	0	0	semantic	0.967
nec	1	0	0	semantic	0.972
nescontroller	1	0	0	structural	0.991
new_component	1	1	0	semantic	0.935
open_free_list	1	0	0	structural	0.000
openfire_top_sim	1	0	0	structural	0.000
pcm_slv_top	1	0	0	structural	0.989
pepelatz_main	1	0	0	semantic	0.697
present_encryptor_top	1	0	0	structural	1.000
ps2_host	0	0	0	structural	0.000

Table 7: Per-strategy reward statistics and pairwise variance analysis ($n = 100$ designs). Cohen's d measures the standardised mean difference. Bonferroni-corrected p-values from Wilcoxon signed-rank tests. Note that population means are similar (Friedman $p = 0.214$); the variance lies in which strategy wins each individual design (Wilcoxon best-vs-worst $p < 0.001$).

Comparison	Mean diff	Cohen's d	W stat	Adj. p
Structural vs. Semantic	-0.038	0.11	1341	1.00
Structural vs. Adversarial	$+0.014$	0.03	1290	0.85
Semantic vs. Adversarial	$+0.052$	0.12	1299	0.91
Friedman: $\chi^2(2) = 3.08$, $p = 0.214$ (population means similar)				
Wilcoxon best-vs-worst per design: $p < 0.001$				

Table 8: Verification quality sub-metrics under LOO evaluation ($n = 100$). Mean pass rate, line coverage, toggle coverage, and mutation kill rate for the strategy selected by each method on each held-out design (best episode per strategy per design). Among deployable methods (all rows below the oracle upper bound), MKR-gated achieves the highest coverage on all four metrics; because it runs a second agent on 34% of designs and keeps the better run, this per-metric dominance over single-run methods is partly a compute effect (see the iso-cost comparison in Section 5.3).

Method	Pass rate ↑	Line cov ↑	Toggle cov ↑	MKR ↑
Oracle (upper bound)	0.927	0.819	0.739	0.726
MKR-gated (ours)	**0.878**	**0.709**	**0.643**	**0.670**
Contextual bandit (route only)	0.861	0.652	0.597	0.617
Fixed-best (semantic)	0.840	0.633	0.577	0.598
LLM-designated rule	0.804	0.646	0.596	0.594
Heuristic rules	0.803	0.622	0.584	0.558
Round-robin mean	0.802	0.605	0.549	0.537

Table 9: Mean reward, line coverage, and MKR for the bandit-selected strategy, grouped by RTL line count. Values are over all designs in that size range. The rows cover the 97 designs with an unambiguous size bucket and a scored bandit-selected episode; the "All" row is the mean over those 97.

RTL lines	Designs	Mean reward	Mean line cov	Mean MKR
< 100	14	0.896	92.1%	76.3%
100–500	48	0.867	84.6%	79.8%
500–3000	18	0.710	64.7%	40.0%
> 3000	17	0.748	76.1%	58.6%
All	97	0.858	82.7%	68.9%

Table 10: MKR threshold analysis: reward spread (best-vs-worst strategy gap per design) for designs below and above different MKR thresholds. Mann-Whitney p-values test whether low-MKR designs have higher spread. The $r_{\text{no-mkr}}$ column replicates at $\tau = 0.5$ on the coverage-only reward (MKR excluded) to rule out circularity: the ratio is essentially unchanged at 2.2× ($p = 0.002$). At $\tau = 0.6$ and $\tau = 0.7$ the low/high groups sum to 99, not 100: one zero-score design has undefined MKR (no testbench mutations to score) and is excluded from those splits.

τ	n_{low}	Spread (low)	n_{high}	Spread (high)	Ratio	p	Ratio ($r_{\text{no-mkr}}$)
0.5	44	0.507	56	0.246	2.1×	0.0022	2.2×
0.6	51	0.509	48	0.211	2.4×	0.0002	n/a
0.7	66	0.503	33	0.090	5.6×	< 0.001	n/a

Table 11: Full method comparison (LOO, $n = 100$, enriched v2 features). Bootstrap 95% CIs from $B = 5000$ samples. MKR-gated triggers on 34/100 (34%) designs.

Method	Mean regret	95% CI	Zero-regret
Oracle	0.000	n/a	100/100
MKR-gated controller (ours)	0.070	$[0.038, 0.108]$	53/100
LINUCB bandit v2 (ours)	0.122	$[0.079, 0.170]$	43/100
Fixed-best (semantic)	0.139	$[0.091, 0.189]$	46/100
Heuristic rules	0.168	$[0.109, 0.235]$	41/100
LLM-designated rule	0.185	$[0.122, 0.247]$	39/100
Round-robin mean	0.173	$[0.140, 0.211]$	7/100
LINUCB bandit v1 (dead feat)	0.131	$[0.079, 0.189]$	45/100

Table 12: Escalation policies under LOO (composite reward, $n = 100$, trigger 34/100, budget-identical to keep-max). Merging the two run testbenches beats keep-max by a small but significant iso-cost margin; the oracle combined suite is a $\sim 6\times$ upper bound (not realizable by selection, and the MKR dimension of it equals the floor on all measurable designs), and the learned complementarity predictor does not beat keep-max.

Escalation policy	Mean regret ↓	95% CI
Bandit (first pick, no escalation)	0.122	$[0.079, 0.169]$
MKR-gated keep-max (main paper)	0.070	$[0.038, 0.108]$
MKR-gated + 2nd, combined suite	0.069	$[0.037, 0.107]$
MKR-gated + predicted-complementary	0.078	$[0.044, 0.120]$
Oracle complementary (ceiling)	0.011	$[0.005, 0.018]$

Table 13: Feature dropout ablation (LOO, $n = 100$, enriched v2 features). Each row zeros one feature from the context vector. `has_fsm` carries the strongest discriminative signal ($\Delta + 0.032$). All new features carry positive signal; compare with v1 where three features were identically zero.

Feature dropped	LOO mean regret	Δ vs. full
None (full model)	0.122	0.000
`has_fsm`	0.154	$+0.032$
`seq_fraction`	0.123	$+0.001$
`rtl_lines_norm`	0.130	$+0.008$
`complexity_norm`	0.128	$+0.006$
`branch_density`	0.124	$+0.002$
`has_protocol`	0.113	-0.009

Table 14: Strategy subset ablation (LOO, $n = 100$). Removing any arm increases regret, validating the three-arm design decision. The semantic+adversarial pair is near-optimal, while structural adds only marginal benefit at current corpus scale.

Strategy subset	LOO mean regret
All 3 strategies (structural + semantic + adversarial)	0.122
Semantic + Adversarial only	0.132
Structural + Adversarial only	0.141
Structural + Semantic only	0.156