TRACESPEC: ACTIVE EVIDENCE ACQUISITION FOR EXECUTABLE MONITOR SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Executable monitor synthesis from language contains two distinct uncertainties: semantic uncertainty about how an alert maps to deployment-local signals, and policy uncertainty about stateful parameters that language alone does not identify. We formulate active monitor synthesis, where a model receives alert language, an opaque local metric/tool catalog, and a measurement budget, and must choose which evidence to acquire before emitting an executable stateful monitor. We introduce TraceSpec, which combines a compact language model for semantic binding and value-of-information decisions with bounded symbolic inference for temporal-policy identification and compilation to MonitorIR. Activation evidence constrains onset-side behavior, whereas recovery evidence resolves parameters that remain observationally equivalent after activation alone. On controlled deployments derived from held-out Thanos and Mimir rules, Qwen3.5-4B reaches 0.904/0.929 execution agreement, versus 0.686/0.738 for the strongest compact controls and 0.912/0.942 for a gold-semantics/oracle-tools upper bound; Phi-4 reaches 0.903/0.922. One informative phase raises execution from about 0.48 to 0.80, and a complementary second phase adds another 10.2–13.9 points. The learned acquisition policy also transfers to an unseen cost value and budget. Within the bounded monitor family studied here, these results show that semantic reasoning, active evidence acquisition, and bounded policy identification form complementary parts of executable synthesis.

1 INTRODUCTION

Operational monitoring rules compress two kinds of knowledge into a small amount of text. The first is semantic: an alert may refer to an error ratio, a saturation condition, or a service-level symptom. The second is deployment-specific: the relevant metric may have a local identifier, useful diagnostic actions may differ across sites, and the exact temporal policy may depend on a threshold, a pending interval, hysteresis, and recovery behavior. Existing systems typically assume these choices are already encoded in a hand-written rule. We study the earlier step: synthesizing an executable monitor when part of the specification must be discovered from the deployment itself.

This setting differs from ordinary text-to-code or text-to-temporal-logic synthesis. Natural language provides the intended behavior, but it does not uniquely determine which local tools instantiate that behavior or which bounded temporal parameters match the site. Measurements can resolve the ambiguity, yet measurements have cost. A useful system must therefore decide *what to observe* as well as *what monitor to write*.

We formulate this problem as **active monitor synthesis**. Each deployment provides an alert description, an opaque metric/tool catalog, descriptions of admissible interventions, a finite credit budget, and a history of observations. The output is an executable state machine. The agent may request measurements before committing to the monitor, but each request consumes credits. This creates a natural division of labor between semantic reasoning and parameter identification.

We introduce **TraceSpec**. A compact language model binds alert intent to deployment-local tools and chooses measurements adaptively. Observed activation and recovery traces are passed to a bounded deterministic verifier, which searches a finite family of temporal policies and compiles the best-fitting candidate to `MonitorIR`. The language model therefore handles the open-ended parts of

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.

the task—semantic transfer and active information acquisition—while deterministic search handles the small structured parameter family.

We evaluate TraceSpec on alert rules from two held-out monitoring repositories, Thanos and Mimir, after training on rules drawn from kube-prometheus, Ceph, and Grafana-derived monitoring mixins. The final evaluation contains 24 Thanos rules and 93 Mimir rules, each paired with four deployment variants. TraceSpec with Qwen3.5-4B reaches 0.904/0.929 execution agreement on the two repositories, while the strongest compact baselines reach 0.686/0.738. Phi-4 reproduces the result at 0.903/0.922. The remaining gap to the gold-semantics/oracle-tools upper bound is small (0.912/0.942).

The active component is equally important. Without measurements, execution remains near 0.48. One informative phase raises it to roughly 0.80; a complementary second phase raises it above 0.90. Under a cost value and budget not used during training, the Qwen policy retains 0.902/0.930 execution. Channel ablations and a restricted RCAEval replay provide additional evidence about which information sources matter and how far the mechanism transfers.

Our contributions are:

1. **A deployment-grounded synthesis problem.** We formalize active monitor synthesis, in which natural-language intent must be bound to opaque local tools and a hidden stateful policy under a measurement budget.
2. **TraceSpec.** We introduce a neuro-symbolic architecture that allocates open-ended semantic reasoning and information acquisition to a language model, then uses bounded deterministic inference to compile an executable monitor.
3. **Complementary-observation analysis.** We make explicit why onset and recovery measurements identify different parameter subsets and show empirically that the second phase adds 10.2–13.9 points beyond the best one-phase regime.
4. **Cross-repository and cross-family evaluation.** Held-out Thanos/Mimir results, a second language-model family, and unseen-cost evaluation show that the method transfers beyond the training repositories and cost values.

2 Related Work

2.1 Natural language to formal specifications and runtime monitors

Large language models have increasingly been used to translate informal requirements into code or formal artifacts. Cohen and Peled (2024) use LLMs to construct runtime verification monitors from natural-language specifications through intermediate temporal-logic representations and self-validation. AutoSpec+ (Wen et al., 2026) uses an LLM together with a symbolic verifier to synthesize proof-relevant program specifications. Related work maps natural language to temporal logic in multiple domains, including NL2TL and Lang2LTL (Chen et al., 2023; Liu et al., 2023). These systems establish that language models and formal checking can be productively combined. TraceSpec differs in the source of uncertainty: the desired monitor is not fully specified by the language. The alert text defines intent, while deployment-local tool bindings and temporal parameters are deliberately hidden and must be identified through interaction.

2.2 Tool use and active information acquisition

Language-model agents can interleave reasoning with actions and tool calls (Yao et al., 2023), and tool-augmented language models can learn to invoke external interfaces (Schick et al., 2023). Runnable tool-use benchmarks and API-grounded models such as API-Bank and Gorilla further test planning and invocation against changing tool catalogs (Li et al., 2023; Patil et al., 2024). Most tool-use benchmarks reward task completion with an already informative tool result. TraceSpec instead makes measurement choice part of the scientific problem: activation and recovery interventions reveal different latent components of the target monitor, calls consume credits, and wrong measurements can leave the hidden policy unidentified. The closest conceptual connection is active experimental design, but our objective is not population estimation or generic exploration; it is synthesis of an executable stateful program from language plus selectively acquired traces.

2.3 Monitoring, alert rules, and executable semantics

Prometheus-style alerting combines an expression with temporal fields such as a pending duration and operational conventions for missingness and recovery (Prometheus Authors, 2026). Systems such as Thanos and Mimir preserve this ecosystem (Thanos Authors, 2026; Grafana Labs, 2026) while changing deployment scale and storage/compute architecture. Existing text-to-query or rule-generation work generally assumes that the relevant metric names and execution semantics are already available. TraceSpec focuses on the complementary problem of identifying which opaque local measurements instantiate the rule and which hidden temporal policy is consistent with observed intervention traces. Root-cause-analysis benchmarks such as OpenRCA and RCAEval evaluate diagnosis from telemetry rather than induction of a new executable monitor (Xu et al., 2025; Pham et al., 2025); newer agent benchmarks broaden planning and failure-diagnosis coverage (Sun et al., 2026; Cai et al., 2026).

2.4 Neuro-symbolic synthesis

A deterministic verifier can prevent a language model from being assigned work that is better solved exactly. This division of labor is central to recent neuro-symbolic specification synthesis (Wen et al., 2026). Our verifier is intentionally narrower: it does not judge unrestricted code generated by the model. Given a semantic plan and selected observations, it enumerates a bounded policy family and returns the executable monitor that best fits the traces. This lets the experiments distinguish semantic/action-policy errors from numeric fitting errors.

3 Problem Formulation

3.1 Deployment instance

A deployment instance contains four observable objects and one hidden target. The observable objects are:

- an alert description x derived from a real monitoring rule;
- a site-local catalog $C = \{c_i\}$ of opaque metric identifiers with natural-language descriptions;
- a tool set $U = \{u_j\}$, where each tool has an opaque identifier, a textual intervention description, a scope, and an integer credit cost;
- a total budget B.

The hidden target is a stateful monitoring policy θ containing a semantic metric/operator choice and temporal parameters. In our controlled task family, the temporal component includes comparator and threshold, pending duration, hysteresis, clear delay, and missing-data behavior. A policy compiled with the correct semantic binding defines a deterministic executable monitor M_θ.

Each deployment contains five opaque metric IDs and eight tools. Four telemetry concepts are present; each concept has one activation/onset intervention and one recovery/missingness intervention. The mapping from semantic concept to identifier is independently salted for every deployment. The tool executor rejects unavailable, repeated, or unaffordable actions and deducts the tool's actual cost from the remaining budget.

3.2 Active interaction

At turn t, the agent observes

$$h_t = (x, C, U, B_t, a_{<t}, o_{<t}),$$

where $a_{<t}$ are previous actions and $o_{<t}$ are their returned traces. It either chooses an admissible tool u_t or emits a semantic plan z. Measurement traces are labeled sequences that reveal only the temporal phase induced by the chosen tool. Activation traces identify onset-side behavior but deliberately omit recovery evidence. Recovery traces start from a coarse abnormal state and reveal hysteresis,

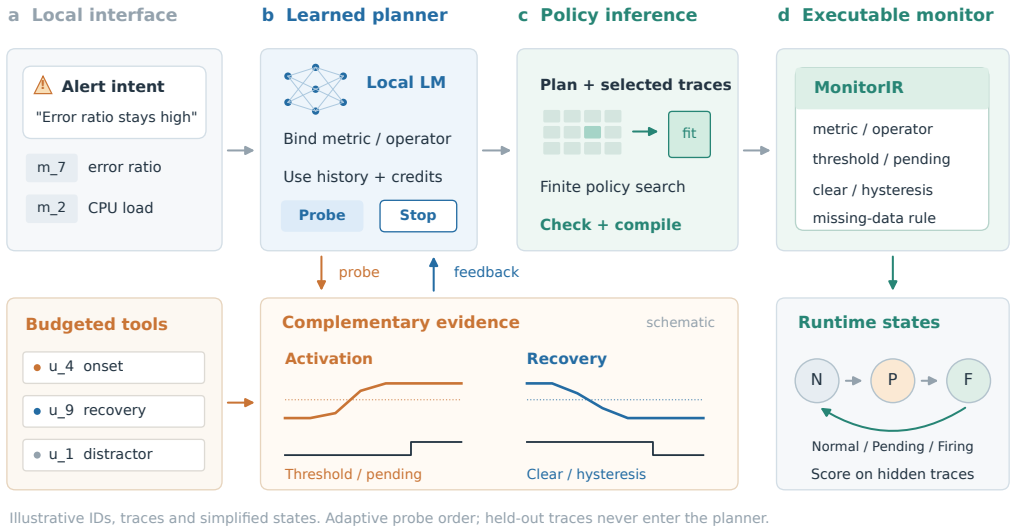

Figure 1: TraceSpec binds alert intent to local metrics and tools, chooses informative measurements from remaining credits and prior observations, and uses a bounded verifier to compile a deterministic MonitorIR. Activation and recovery observations identify complementary policy parameters. IDs, traces, and abbreviated states are schematic, not measured deployment episodes; the planner is not required to follow a fixed probe order.

clear delay, and missing-data behavior. Wrong-concept tools contain nontrivial ramps, plateaus, and outages while the target state remains inactive, so variance or "interesting-looking trace" heuristics do not identify the right tool.

The goal is to maximize agreement between the compiled monitor and hidden test traces while obeying the credit budget. Hidden traces test the exact comparator/threshold, pending behavior, clear boundary, hysteresis, and missingness and are never shown to the model.

3.3 Evaluation budgets

We report three normal budgets. **B0** gives no active measurement and tests language/tool binding alone. **B1** permits one informative call. **B2** permits a complementary second call. The design makes the two phases intentionally nonredundant: activation evidence is sufficient for onset parameters but underdetermines recovery parameters; recovery evidence supplies the missing information. The post-hoc best singleton remains below the two-phase oracle, providing a direct check that B2 is not merely "more tokens."

Structural complementarity. Write the bounded temporal policy as $\theta = (\theta_{\text{on}}, \theta_{\text{off}})$, where θ_{on} contains comparator/threshold and pending behavior and θ_{off} contains hysteresis, clear delay, and missing-data behavior. An activation-only history terminates before recovery-side transitions occur, so distinct θ_{off} values are observationally equivalent under that history. A recovery intervention breaks this equivalence by exposing the return-to-normal transitions. The two phases are therefore complementary experiments rather than repeated context. This statement is specific to the bounded monitor family used here.

4 TraceSpec

4.1 Semantic planner

TraceSpec fine-tunes a compact local language model with a strict structured-action interface. At every turn the model receives the alert text, local catalog, currently admissible tools and costs, remaining budget, and previous observations. The output grammar permits either a tool action or a

final semantic plan. Tool IDs themselves have no stable semantics across deployments, forcing the model to use the descriptions and observed outcomes.

The plan captures the semantic quantities needed by the verifier: which metric/operator the alert refers to, which direction and temporal phase are relevant, and which observed traces should constrain onset and recovery behavior. It does not directly emit unrestricted monitoring code or arbitrary numeric parameters.

4.2 Training task construction

Development begins from real monitoring rules and augments them with independently sampled deployment-local hidden policies. The v8.2 training corpus contains 120 training rules and 632 deployment episodes. Multi-turn trajectories, cost conditions, and two bounded off-policy histories yield 9,480 policy views. The off-policy histories present either a recovery measurement before activation or a distractor measurement first; the correct next action must recover by selecting the missing informative phase. This trains bounded replanning without reinforcement learning or long-horizon rollout optimization.

Tool-description paraphrase banks are split so that structural/source evaluation does not reuse the training phrasing. Training costs are limited to values 1 and 2 and budgets 2 and 3. Cost extrapolation later introduces cost 3 and budget 4.

4.3 Cost-aware action interface

The executor exposes only currently admissible tools. A tool is removed when it has already been used, is unavailable in the deployment, or costs more credits than remain. This seemingly small interface constraint matters: an earlier formulation allowed the language model to repeatedly request an already-invalid tool and then relied on downstream rejection. The final interface makes the remaining action set truthful while preserving the underlying decision problem.

TraceSpec is not given the oracle trajectory. The model must choose whether the currently available evidence is sufficient and, when another measurement is useful, select the semantically complementary intervention rather than an arbitrary second call.

4.4 Deterministic verifier and MonitorIR

Once the model emits a semantic plan, a bounded verifier fits the numeric policy. It enumerates the allowed comparator, threshold, pending, hysteresis, clear-delay, and missing-data choices in a staged search, selecting the best fit to the observed states and transitions. The selected candidate is compiled into `MonitorIR`, a deterministic executable state machine used for all scoring.

This separation is deliberate. It keeps the language model responsible for the components that require cross-source semantics—binding alert intent to local metrics, choosing interventions, and interpreting observation history—while deterministic finite search handles numeric policy identification. A gold-semantics + oracle-tools configuration therefore provides an informative ceiling rather than a competing deployable method.

4.5 MonitorIR as an executable interface

The output of TraceSpec is deliberately smaller than a general program. A `MonitorIR` instance specifies the observed metric, comparison direction, threshold, pending duration, hysteresis, clear delay, and missing-data behavior. These fields define a deterministic state machine with inactive, pending, and firing states. This representation gives the language model a concrete target without asking it to reproduce repository-specific syntax. It also makes the same synthesized policy executable under every deployment variant used in our evaluation.

This separation is useful beyond implementation convenience. Natural-language alert descriptions often determine *what* should be monitored but do not uniquely determine numerical timing behavior. Conversely, short intervention traces are informative about timing and recovery but do not name the intended metric or operator. TraceSpec therefore assigns the two information sources to complementary parts of the pipeline: language and catalog descriptions establish semantic identity, while active

observations identify the small remaining policy family. The bounded verifier combines them only after the planner has selected the relevant evidence.

4.6 Model training

The primary model is Qwen3.5-4B trained with BF16 rank-16 LoRA. Three seeds (2/3/7) are used for final reporting. Phi-4 provides a second model family; Mistral-7B is retained as a qualified negative because it learns diagnosis and paired-cost behavior but remains weaker on exact single-cost stopping. Training is local and bounded: each Qwen seed takes 3.12–3.43 h in the final configuration, and Phi-4 takes 7.05 h. Full implementation and artifact details are deferred to the appendix.

5 Experimental Setup

5.1 Repositories and task instances

Training uses 120 monitoring rules drawn from kube-prometheus, Ceph, and Grafana monitoring mixins. Evaluation uses 24 Thanos rules and 93 Mimir rules extracted from pinned repository revisions, with four deployment variants per rule (96 and 372 instances). The repositories differ in alert vocabulary, operator structure, and metric composition, providing a source-level test of semantic transfer. Deployment variants use independently sampled local tool identifiers and bounded temporal policies. The alert language and monitoring operators come from the real repositories; the private temporal policies and intervention traces are generated by the controlled task simulator. Appendix A documents source separation, extraction, and inclusion details.

5.2 Methods and baselines

We compare TraceSpec with a fixed lexical policy, a compact Naive Bayes baseline, MiniLM sentence representations with learned heads, and a gold-semantics/oracle-tools upper bound. Qwen3.5-4B is the primary TraceSpec planner and is reported over three seeds; Phi-4 provides a second model family. For cost experiments, compact baselines are paired with text-derived schedulers so that improvements cannot be attributed merely to the presence of an action scheduler.

5.3 Metrics

The primary metric is **execution agreement**: the fraction of correctly predicted hidden trace time steps within each deployment, averaged across deployments. We additionally report probe calls and complementary-probe selection; the result artifacts include balanced accuracy, transition F1, episode-exact rate, parse success, and policy violations. Final method comparisons use paired bootstrap intervals clustered by source rule.

5.4 Budget and transfer settings

Training uses costs 1/2 and budgets 2/3. Evaluation includes the corresponding seen-cost conditions and an unseen profile containing cost 3 and budget 4. We also evaluate a restricted onset-only replay on RCAEval fault-injection traces. Full task-construction and replay details are given in the appendix.

5.5 Ablation settings

To understand which information sources support synthesis, we perturb alert descriptions, catalog descriptions, probe descriptions, probe scope, returned traces, and probe labels on the source-OOD development set. These are reported as ablations rather than additional headline tasks.

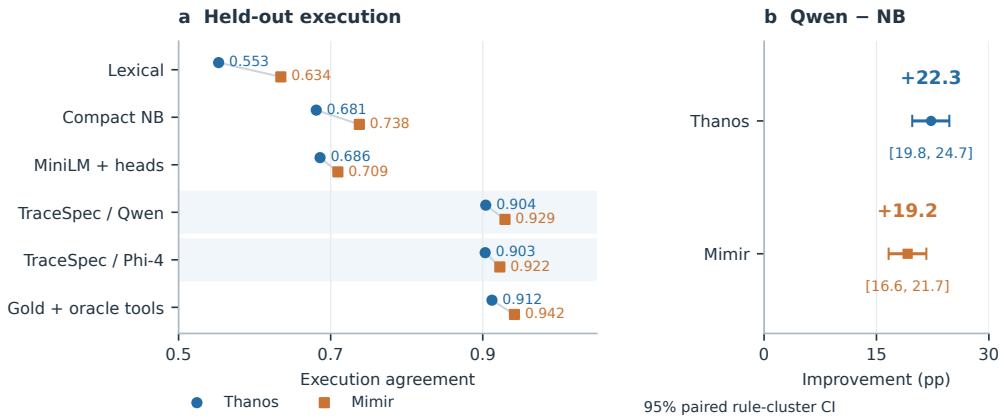

Figure 2: Held-out evaluation. (a) Execution agreement on Thanos (96 deployments) and Mimir (372 deployments); Qwen/MiniLM aggregate three seeds, while other rows are single reported runs. The oracle-tools condition is privileged, not a deployable baseline. (b) Qwen improvements over a common Compact NB reference, with archived rule-cluster paired-bootstrap 95% intervals (5,000 resamples). These differences use NB for both repositories and should not be confused with improvement over the best compact baseline on each source.

Table 1: Held-out execution agreement. Qwen and MiniLM average three seeds. Other rows are single reported runs.

Method	Thanos	Mimir
Lexical policy	0.553	0.634
Compact NB	0.681	0.738
MiniLM + heads	0.686	0.709
TraceSpec Qwen3.5-4B	**0.904**	**0.929**
TraceSpec Phi-4	0.903	0.922
Gold semantics + oracle tools	0.912	0.942

6 Results

6.1 TraceSpec closes most of the gap to oracle semantics

Figure 2 summarizes the main results. On Thanos, TraceSpec with Qwen3.5-4B reaches **0.904** execution agreement, compared with MiniLM at 0.686 averaged over three seeds and 0.912 for gold semantics with oracle tools. On Mimir, the corresponding values are **0.929**, 0.738, and 0.942. Phi-4 independently reaches 0.903 and 0.922. The improvement over compact controls is large on both repositories. Thanos MiniLM reaches 0.688 in its strongest seed (0.686 over three seeds), while Compact NB at 0.738 is the strongest compact Mimir average. The small distance to the privileged upper bound is specific to this bounded policy family.

The result is not specific to a single planner family. Qwen and Phi-4 obtain nearly identical execution on Thanos and differ by less than one point on Mimir. The oracle comparison also shows that the task is not saturated by the deterministic verifier alone: compact semantic baselines remain substantially below both language-model planners even though they share the same downstream compiler.

6.2 Active measurements account for most of the remaining gain

The normal budget curve gives the clearest picture of the active component. With no deployment measurement (B0), execution is 0.486 on Thanos and 0.483 on Mimir. One informative phase (B1) raises agreement to 0.802 and 0.790. Allowing the complementary activation/recovery pair (B2)

Table 2: Budgeted active measurement with Qwen3.5-4B under unit-cost tools (budgets 0/1/2), averaged over three seeds. Calls are averages per deployment; pair rate is the fraction of episodes using complementary activation and recovery evidence.

Source	Budget setting	Exec.	Calls	Pair rate
Thanos	B0: no measurement	0.486	0.00	–
Thanos	B1: one phase	0.802	1.00	0.000
Thanos	B2: two phases	0.904	2.00	0.990
Mimir	B0: no measurement	0.483	0.00	–
Mimir	B1: one phase	0.790	1.00	0.000
Mimir	B2: two phases	0.929	2.00	0.967

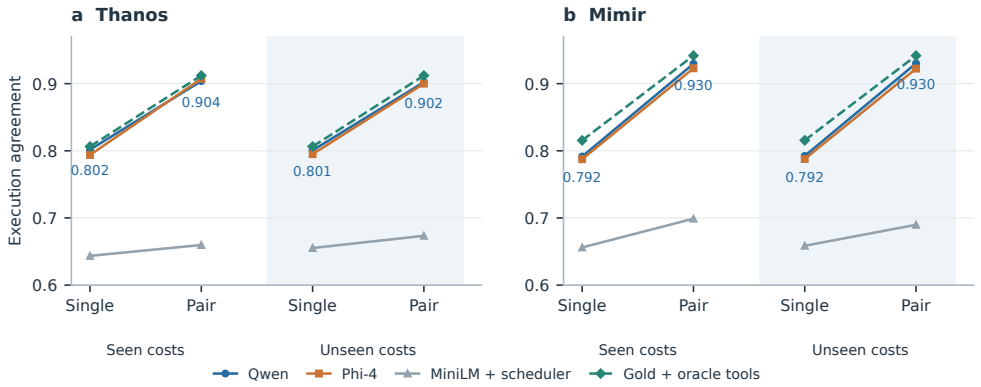

Figure 3: Acquisition under heterogeneous seen and unseen credit profiles, distinct from the unit-cost setting in Table 2. The Qwen policy uses affordable single or complementary paired evidence on each held-out repository. Shaded areas identify the unseen cost-3, budget-4 configuration. Lines connect settings within the same profile only; the result is not a measured latency or dollar-cost Pareto frontier.

reaches the final 0.904 and 0.929. The second observation therefore provides an additional 10.2–13.9 points after the large first-step gain.

The learned policy usually spends this second call on complementary evidence rather than repeating the first phase: under paired-cost conditions, Qwen selects an activation/recovery pair on 0.983 of Thanos deployments and 0.966 of Mimir deployments; the normal-budget B2 table reports its separate setting. A post-hoc best-singleton oracle remains below the two-phase oracle, so the gain is not explained by merely choosing a better first probe.

6.3 Cost behavior transfers to unseen credit values

Under an unseen profile containing cost 3 and budget 4, Qwen reaches 0.902 on Thanos and 0.930 on Mimir with roughly two calls per episode, essentially matching the seen paired-cost setting. Phi-4 shows the same qualitative behavior. We interpret this as transfer over normalized credit arithmetic, not as evidence about real latency or monetary cost.

6.4 Ablations and external replay

Supplementary Fig. S1 summarizes dependence on each major information channel in source-OOD development. Qwen drops from 0.938 on the source-OOD development set to 0.807 when alert descriptions are shuffled, 0.698 when catalog meanings are shuffled, 0.623 when probe descriptions are shuffled, 0.512 when probe scope is hidden, 0.567 when returned traces are shuffled, and 0.500 when probe labels are reversed. Phi-4 follows the same qualitative pattern. These ablations support the intended decomposition: the planner uses both semantic descriptions and returned observations.

Training-rule scaling is positive in aggregate but non-monotonic: external-OOD execution rises from 0.874 with 15 rules to 0.924 with 120, while structural OOD stays near 0.91–0.93. On a restricted RCAEval onset-only replay, the method reaches 0.968 execution over 40 hidden repetitions. This replay covers only onset calibration, so we treat it as a transfer check rather than a full production-monitoring evaluation.

6.5 Efficiency and error structure

The method remains small enough for local deployment. Final greedy inference takes 2.86–2.95 seconds per deployment for Qwen3.5-4B and 3.16–3.26 seconds for Phi-4 when the model is already loaded. The three Qwen LoRA runs require 3.12–3.43 hours each; Phi-4 requires 7.05 hours under the same task recipe. These measurements are not optimized systems benchmarks, but they show that the active loop does not require a large frontier model.

Errors can enter at two qualitatively different stages. The planner may bind the wrong metric/tool or request an uninformative observation; alternatively, the correct semantic plan may fall outside the bounded temporal family searched by the verifier. This decomposition is useful operationally because the two failure modes can be inspected separately. The remaining gap to the privileged oracle configuration does not identify which fraction of errors originates in semantic planning, policy-family misspecification, or execution; a module-level error attribution requires an additional controlled study.

7 Discussion

TraceSpec addresses a bounded but nontrivial regime: an alert identifies the monitoring intent, while local tool bindings and stateful policy values are unknown and must be learned through budgeted observations. Language models handle semantic transfer and measurement choice; a deterministic verifier fits the finite policy family and compiles executable MonitorIR. Activation and recovery traces resolve complementary uncertainties rather than merely supplying more context.

Our experiments use real alert language from held-out repositories but controlled deployment-private policies and simulated interventions; RCAEval checks only onset-side offline replay. The unseen-credit experiment demonstrates transfer under normalized integer costs, not operational latency or risk. Moreover, the privileged oracle gap does not identify which internal stage contributes the remaining errors, nor do these results isolate an advantage over a same-model fixed complementary two-probe policy. Live or sandboxed deployment with observed recovery behavior and measured intervention cost remains open.

For fully specified rules, passive text-to-specification methods may be preferable; for unbounded stateful behavior, finite policy search is insufficient. The task-regime comparison, efficiency implications, and extended discussion appear in the supplement.

Boundaries of the evidence. The held-out repository text is authentic, but hidden policies and intervention responses are controlled. These experiments establish evidence complementarity under a bounded interface, not production reliability or a universal benefit over a same-model fixed-two-probe policy. Wrong tool bindings, unobserved recovery behavior, and out-of-family targets remain open failure modes. Operational validation will require recovery-side replay, real intervention costs, and missing-telemetry tests.

8 Conclusion

We introduced active monitor synthesis and TraceSpec, combining language-model semantic binding and budgeted information acquisition with bounded deterministic policy fitting. Across held-out Thanos and Mimir repositories, two model families substantially outperform compact baselines and approach a gold-semantics/oracle-tools upper bound. Complementary observations provide large gains, while ablations show that both semantic descriptions and returned traces matter.

AI USE STATEMENT

We used generative AI tools during research ideation, literature triage, code generation and debugging, analysis of experimental artifacts, and manuscript drafting and editing. All experimental results reported in this paper were produced by executable code and checked against frozen result artifacts; AI-generated code and prose were reviewed and revised by the authors. We take responsibility for the final content, claims, and artifacts in this submission.

REPRODUCIBILITY STATEMENT

The task construction, action executor, bounded verifier, source extractors, evaluation scripts, frozen result tables, statistical summaries, and artifact hashes are documented in the supplementary material. Appendix A describes final-source isolation, Appendix B gives training details, and Appendix C reports additional paired statistics. We will provide an anonymized code package with the submission.

REFERENCES

M. Abdin et al. Phi-4 Technical Report. arXiv:2412.08905, 2024.

Y. Cai et al. A Multi-Dataset Benchmark for Evaluating LLM Agents in Microservice Failure Diagnosis. arXiv:2606.29193, 2026.

I. Cohen and D. Peled. End-to-End AI Generated Runtime Verification from Natural Language Specification. In *AISoLA*, LNCS 14129, pp. 362–384, 2024.

Grafana Labs. Grafana Mimir: horizontally scalable, highly available, multi-tenant time series database. Project documentation, 2026.

L. Pham, H. Zhang, H. Ha, F. Salim, and X. Zhang. RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems with Telemetry Data. In *Companion Proceedings of the ACM Web Conference*, pp. 777–780, 2025.

Qwen Team. Qwen3.5-4B Model Card. Hugging Face model release, 2026.

T. Schick et al. Toolformer: Language Models Can Teach Themselves to Use Tools. In *NeurIPS*, 2023.

H. Sun et al. Agent Planning Benchmark: A Diagnostic Framework for Planning Capabilities in LLM Agents. arXiv:2606.04874, 2026.

Thanos Authors. Thanos: Highly Available Prometheus Setup with Long Term Storage Capabilities. Project documentation, 2026.

C. Wen et al. Formally Specifying the Intended Behavior of the Program: LLM-Driven Neuro-Symbolic Program Specification Synthesis. In *ACL System Demonstrations*, pp. 672–682, 2026.

J. Xu et al. OpenRCA: Can Large Language Models Locate the Root Cause of Software Failures? In *ICLR*, 2025.

S. Yao et al. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*, 2023.

Y. Chen, R. Gandhi, Y. Zhang, and C. Fan. NL2TL: Transforming Natural Languages to Temporal Logics using Large Language Models. In *EMNLP*, 2023.

M. Li, Y. Zhao, B. Yu, F. Song, H. Li, H. Yu, Z. Li, F. Huang, and Y. Li. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *EMNLP*, pp. 3102–3116, 2023.

J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah. Grounding Complex Natural Language Commands for Temporal Tasks in Unseen Environments. In *CoRL*, PMLR 229:1084–1110, 2023.

S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs. In *NeurIPS*, 2024.

Prometheus Authors. Defining Alerting Rules. Prometheus project documentation, 2026.