FAROS: CHARACTERIZING CODING-AGENT SEARCH FOR EXECUTABLE TIME-SERIES ANOMALY DETECTORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents can construct executable anomaly detectors from labeled time-series data. However, their open-ended search can vary substantially across runs in both length and resulting detector quality, making systematic evaluation difficult. We introduce FAROS (Framework for Agentic Run Observation and Synthesis), an evaluation framework that records search states during ongoing coding-agent search and applies a detector synthesis procedure to obtain detector readouts from selected states. This enables comparison of detector quality at selected points during search. It also separates variability across independent search states from variability in repeated synthesis from a fixed state. Using FAROS, we study detector construction from few labeled anomaly events and how reasoning setting and search effort affect detector quality and repeatability. Our experiments yield three findings. First, coding agents can construct detectors competitive with specialized pretrained time-series anomaly detectors from few labeled anomaly events. Second, at larger common output-token targets, stronger reasoning achieves higher mean detection performance, while longer lower-effort search does not recover the same performance. Third, we observe contrasting changes in search-state and synthesis variability across reasoning settings and the two tested label budgets as search proceeds. At the largest output target, search-state variability remains the larger component across reasoning settings.

1 INTRODUCTION

What constitutes an anomaly can vary substantially across monitored time series, requiring detectors to be adapted to individual targets (Chandola et al., 2009; Carmona et al., 2022). A sharp spike may indicate an anomaly in one series while occurring as part of normal behavior in another. However, the labeled anomaly events needed for such adaptation are often scarce (Pang et al., 2019). Moreover, selecting appropriate detection principles and configurations for heterogeneous targets can require substantial expertise (Blázquez-García et al., 2021; Schmidl et al., 2022; Sylligardos et al., 2023).

Recent work has shown that LLMs can recognize time-series anomalies (Zhou & Yu, 2025; Liu et al., 2025; Park et al., 2025) and that LLM agents can generate executable anomaly-detection rules (Gu et al., 2025). These results suggest the possibility of using coding agents to automatically analyze labeled time-series data and construct detectors adapted to individual targets.

Systematically evaluating such detector construction is challenging because coding-agent search is open-ended. Given the same task, runs can differ substantially in their number of interactions, generated output, and sequence of candidate detectors and revisions. Endpoint-only evaluation therefore compares runs after potentially different amounts of search and different construction trajectories, making it difficult to characterize how detector quality changes with search effort or to compare run-to-run variability at matched stages of construction.

We introduce FAROS (Framework for Agentic Run Observation and Synthesis), an evaluation framework that records selected states during ongoing coding-agent search (Figure 1). Each recorded state contains the agent's conversation history and workspace files, and FAROS independently synthesizes executable detectors from these states. This separates ongoing search from evaluation and allows states to be selected according to a chosen measure of search progress without terminating

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.

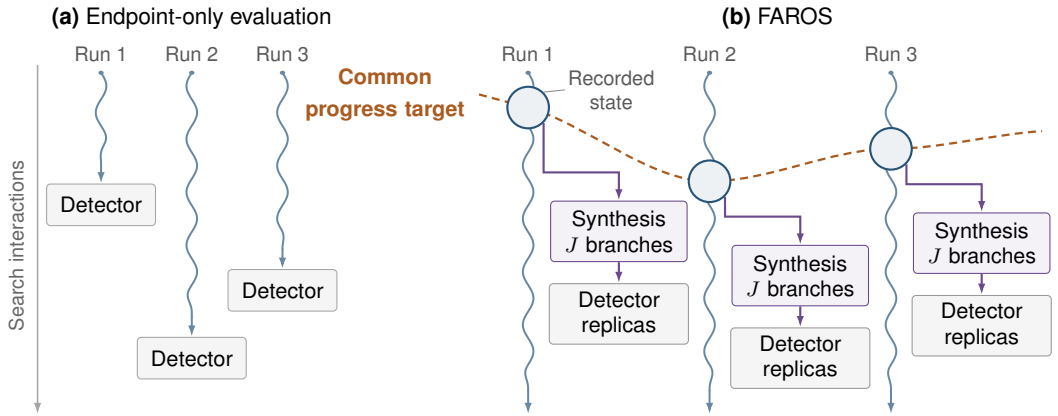


Figure 1: Endpoint-only and FAROS evaluation of open-ended coding-agent search. (a) Endpoint-only evaluation observes a detector when each run terminates, leading to comparisons after different amounts of search. (b) FAROS records states when each run reaches a common progress target while search continues, enabling detector quality to be compared at matched points across runs. Repeated detector synthesis from each recorded state further separates performance variation across independent searches from variation introduced during detector synthesis.

or redirecting the underlying search. For comparison across runs, states can be aligned using a common progress measure such as cumulative output tokens. Repeated synthesis from the same state separates variability across independently reached construction states from variability introduced during detector synthesis.

Using FAROS, we study two questions about coding-agent detector construction. First, given the scarcity of labeled anomaly events available for adaptation, how well can coding agents construct useful executable detectors from a few labeled anomaly events (**RQ1**)? Second, how do reasoning setting and search effort affect detector quality and repeatability across independent runs (**RQ2**)?

Our contributions are both methodological and empirical. Methodologically, FAROS enables open-ended coding-agent searches to be aligned at common progress targets and decomposes repeatability into variability across independently reached search states and variability in synthesis from a fixed state. Empirically, we obtain three main findings. First, coding agents can construct executable anomaly detectors competitive with specialized pretrained time-series models from few labeled anomaly events. Second, stronger reasoning reaches higher detector quality at larger common output targets, and longer lower-effort search does not recover the same performance under the tested budgets. Third, search-state and synthesis variability change differently as search proceeds, with contrasting patterns across different label budgets. At the largest output target, search-state variability remains the larger component across all reasoning settings at both label budgets. Together, these findings show that FAROS reveals how detector quality and distinct sources of variability evolve during ongoing coding-agent search. These aspects are not resolved by endpoint-only evaluation.

2 Related Work

Conventional time-series anomaly detection Time-series anomaly detection encompasses a broad range of statistical, machine-learning, and deep-learning approaches, whose effectiveness varies across time series and anomaly types (Blázquez-García et al., 2021; Schmidl et al., 2022; Sylligardos et al., 2023). Consequently, the appropriate detection principle and model configuration depend on the characteristics of the target data. Model selection and hyperparameter optimization can automate part of this adaptation, but they operate within a user-defined candidate space of models and hyperparameters (Sylligardos et al., 2023; Akiba et al., 2019). In contrast, our setting uses a coding agent to construct the detector implementation itself, including feature computation and scoring logic.

LLMs and agents for anomaly detection Recent work has applied LLMs and LLM agents to time-series anomaly detection in several ways. AnomLLM evaluates zero- and few-shot anomaly

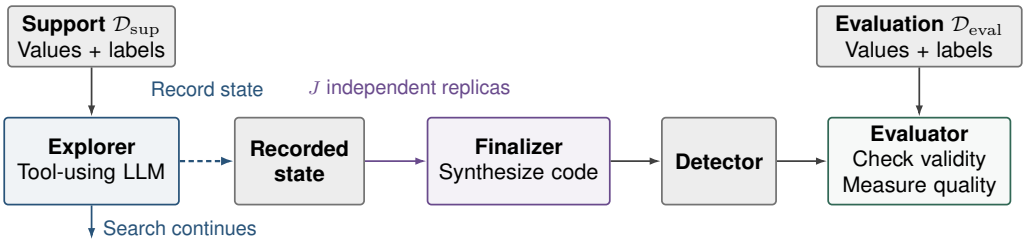


Figure 2: FAROS workflow. The Explorer searches using support-set feedback while states $S_B = (H_B, W_B)$ are recorded at selected response boundaries without interrupting the search. Each recorded state seeds J independent synthesis branches, each allowing up to A attempts to produce a valid executable detector. The non-LLM Evaluator validates generated programs on the support set and evaluates valid detectors on held-out evaluation data. Synthesis and evaluation results are not returned to the Explorer.

recognition from textual and visual time-series representations (Zhou & Yu, 2025). LLMAD combines retrieval of normal and anomalous examples with anomaly-specific reasoning for in-context detection (Liu et al., 2025), while AnomaMind uses tool interactions to gather evidence and iteratively refine anomaly decisions (Tao et al., 2026). Other work uses LLMs to construct detection logic. Argos generates and refines executable Python anomaly rules through multiple agents (Gu et al., 2025), while Zhang & Jain (2026) use LLMs to generate and iteratively refine symbolic detection rules from domain-informed labels. These approaches construct anomaly detectors or detection rules, whereas FAROS is an evaluation framework for such agentic construction procedures. FAROS provides a general evaluation layer for compatible tool-using construction procedures, recording recoverable agent states and applying a common state-conditioned synthesis readout while keeping evaluation isolated from the continuing search.

Agent trajectories and intermediate states Recent work has examined LLM-agent behavior beyond final task outcomes. AgentLens assesses software-engineering agent trajectories beyond final task success, characterizing process quality across exploration, implementation, and verification (Sahoo et al., 2026). DIVERT snapshots full agent–environment states and resumes interaction from selected junctions, branching alternative user responses to improve behavioral coverage and failure discovery (Nakash et al., 2026). Related work also uses intermediate natural-language reasoning before program generation. Hypothesis Search generates and tests programs from natural-language hypotheses (Wang et al., 2024), while PlanSearch searches over natural-language plans to increase solution diversity and code-generation performance (Wang et al., 2025a). These methods use trajectories or intermediate reasoning to characterize behavior, expand behavioral coverage, or improve generation. FAROS instead uses recorded states from an ongoing coding-agent search as controlled evaluation points, enabling comparisons at matched search progress and decomposition of variability across independent searches and final synthesis.

3 FAROS FOR EVALUATING CODING-AGENT DETECTOR CONSTRUCTION

FAROS separates three components (Figure 2). The *Explorer* is a tool-using LLM agent that analyzes labeled data and tests candidate detectors. The *Finalizer* is an LLM invoked independently from a recorded Explorer state to produce an executable detector. The deterministic, non-LLM *Evaluator* executes the generated program, checks its execution contract, and measures detection performance.

3.1 TASK FORMULATION

We consider a time-series anomaly-detection task with a labeled support set $\mathcal{D}_{\text{sup}}$ and a separate held-out evaluation set $\mathcal{D}_{\text{eval}}$. For a task with d observed variables, the support set is

$$\mathcal{D}_{\text{sup}} = \{(x_i, y_i)\}_{i=1}^{K}, \qquad x_i \in \mathbb{R}^{T_i \times d}, \quad y_i \in \{0,1\}^{T_i}, \tag{1}$$

where T_i is the number of timestamps in series i and $y_{it} = 1$ indicates that timestamp t is anomalous. Here, K denotes the number of support series. In the controlled synthetic experiments, each support series contains exactly one anomaly event, making K equal to the labeled-event budget.

Given $\mathcal{D}_{\text{sup}}$, the coding agent constructs an anomaly-scoring function. For an input series $x \in \mathbb{R}^{T \times d}$, the detector returns $f(x) \in \mathbb{R}^T$, with one finite real-valued anomaly score per timestamp, where larger values indicate stronger anomaly evidence. The resulting detector is an executable numerical program that requires no LLM at deployment and is restricted to standard numerical-computing libraries. Full execution constraints are given in Appendix B.

The held-out evaluation set is

$$\mathcal{D}_{\text{eval}} = \{(x_j^E, y_j^E)\}_{j=1}^N. \tag{2}$$

Neither $\mathcal{D}_{\text{eval}}$ nor its evaluation results are available to the Explorer or Finalizer. The Evaluator accesses $\mathcal{D}_{\text{eval}}$ only after a detector has been frozen.

3.2 Agentic exploration and recorded states

The Explorer interacts with the labeled support set through OpenHands (Wang et al., 2025b) using terminal, file-editor, and Think tools. The Explorer's finish tool is disabled to maintain continuing search under an externally imposed budget. The Explorer can use terminal commands to read the support values and labels from files and analyze them by executing Python code. It can also implement and test candidate detectors using a provided scorer that evaluates detection performance on the support set. A workspace directory is available for creating and modifying code and analysis files during exploration. Filenames and column names that reveal the original data source or anomaly type are replaced with neutral identifiers before the data are presented to the Explorer.

We index exploration by B, the number of completed Explorer responses. At selected response boundaries, we record a state

$$S_B = (H_B, W_B), \tag{3}$$

where H_B is the saved conversation history and W_B is the saved workspace. The Explorer uses H_B as context and accesses W_B through its tools. After recording, the same Explorer continues until the prescribed run limit is reached. At $B = 0$, the recorded history contains the task instructions, but the Explorer has not yet observed any support values or labels. Full prompts and recording semantics are given in Appendix B.

Although recorded states are indexed by the completed-response count B, analyses may select states using other measures of search progress. Let O_B denote cumulative Explorer output tokens through response B. For an output target τ, we select the first completed response satisfying $O_B \geq \tau$.

3.3 State-conditioned detector synthesis

For each recorded state S_B, we launch J independent synthesis branches. Each branch produces at most one detector replica and permits up to A fresh attempts. The workspace W_B is copied into the branch, but the Finalizer receives only the serialized history H_B as model context and is not allowed to read additional files or execute code. Because H_B contains earlier messages, tool calls, and returned observations, code written through file-editor actions or terminal commands remains available to the Finalizer. In each attempt, the Finalizer must create the deployment program with a single file-editor action in one completed model response.

The Evaluator checks each generated program on the support set for execution validity, including resource limits and one finite score per timestamp. The first valid program is frozen and evaluated unchanged on the evaluation set. If generation or support validation fails, a fresh attempt starts from the same state without information from earlier attempts. A replica is unsuccessful if all A attempts fail or if the frozen program later violates the execution contract on the evaluation set. Results from synthesis branches are not returned to the continuing Explorer search.

4 Experimental setup

4.1 Datasets

Synthetic dataset We generate a synthetic dataset using the AnomLLM generator (Zhou & Yu, 2025). The dataset contains eight anomaly types: range, point, frequency, trend, flat-trend, noisy-point, noisy-frequency, and noisy-trend. For each anomaly type, we generate 400 series for the

training split and 400 series for the evaluation split, all univariate and of length 1,000. Training and evaluation series are generated with distinct seeds. From the training split of each anomaly type, we construct three independently sampled support draws with nested budgets $K \in \{1, 2, 4, 8, 16\}$. Each selected support series contains exactly one anomaly event, making K both the number of support series and labeled anomaly events. After excluding normal-only series, the held-out evaluation sets contain 2,065 anomaly-containing series in total, including 919 with multiple events. Full split construction is given in Appendix A.

TSB-AD-U We additionally evaluate on TSB-AD-U (Liu & Paparrizos, 2024), a heterogeneous time-series anomaly-detection benchmark including real-world data. We construct 27 cases from its tuning data and evaluate the resulting detectors on 343 held-out series. Each case contains between one and five labeled support series. Full case construction is given in Appendix A.

4.2 FAROS CONFIGURATION

The Explorer runs through OpenHands in a Docker environment. The Evaluator executes generated detectors in an isolated environment with a 120-second time limit per series. Full prompts, access restrictions, and runtime details are provided in Appendix B. We use `gpt-5.6-luna` (Luna) and `Qwen3.8-27B` (Qwen) as Explorer backbones. The Finalizer is fixed to Luna with `xhigh` reasoning for both backbones. From each recorded state included in an analysis, we launch $J = 3$ independent synthesis branches, each allowing up to $A = 3$ attempts.

4.3 RQ1: DETECTOR CONSTRUCTION FROM FEW LABELED EVENTS

RQ1 evaluates how well coding agents can construct anomaly detectors from limited labeled supervision. On the synthetic dataset, we evaluate Luna and Qwen Explorers with `xhigh` reasoning at $K \in \{1, 2, 4, 8, 16\}$. For each anomaly type, labeled-event budget, support draw, and Explorer backbone, we run three independent Explorer searches up to $B = 50$ and synthesize detectors from the endpoint state. We additionally evaluate the same construction procedure on TSB-AD-U.

We compare FAROS with non-neural supervised classifiers (CART (Breiman et al., 1984), logistic regression, Random Forest (Breiman, 2001), Extra Trees (Geurts et al., 2006), histogram gradient boosting, and XGBoost (Chen & Guestrin, 2016)), neural models trained from scratch (FCN (Wang et al., 2017), TCN (Bai et al., 2018), BiLSTM (Hochreiter & Schmidhuber, 1997; Schuster & Paliwal, 1997), and Transformer (Vaswani et al., 2017)), and pretrained time-series models (MOMENT (Goswami et al., 2024), TSPulse (Ekambaram et al., 2026), and Time-RCD (Lan et al., 2026)) in zero-shot and fine-tuned settings. Full configurations are provided in Appendix E.

As a workflow-level comparison, we additionally evaluate a standard coding agent baseline using Luna with `xhigh` reasoning. It uses the same OpenHands framework, support data, tools, and library restrictions as FAROS. The agent directly produces its final detector at natural termination without separate state-conditioned synthesis.

4.4 RQ2: REASONING, SEARCH EFFORT, AND REPEATABILITY

RQ2 studies how reasoning setting and search effort affect detector quality and repeatability. We compare five Luna reasoning settings, `none`, `low`, `medium`, `high`, and `xhigh`, at $K \in \{1, 16\}$ across six cumulative Explorer-output token targets, $\tau \in \{5, 10, 15, 20, 25, 30\}$k. For each target, we select the first completed response whose cumulative output reaches or exceeds the target. Cumulative output counts all tokens generated from the start of the search, including reasoning tokens. At each label budget, we evaluate eight anomaly types and three support draws per type, with five independent Explorer searches per reasoning setting and three detector syntheses from each selected state.

Performance summaries use valid detector outcomes. For the variance analysis, we retain only Explorer runs with valid synthesis outcomes at all six output targets. This keeps the same Explorer runs across targets. We estimate synthesis variance, $\widehat{V}_F$, from variation among syntheses from the same state and search-state variance, $\widehat{V}_E$, from variation across independent searches after accounting for synthesis variance in each run-level mean. The variance of a single synthesized detector is estimated as $\widehat{V}_E + \widehat{V}_F$. Full estimator definitions are provided in Appendix D.

Table 1: AP and VUS-PR on the synthetic dataset by labeled-event budget K, and AP on TSB-AD-U. FAROS results use the endpoint state at $B = 50$. Both Explorer backbones use `xhigh` reasoning, and the Finalizer is Luna with `xhigh` reasoning. The standard coding agent uses Luna `xhigh` and returns its detector at natural termination without a separate Finalizer. Bold indicates the highest mean in each column.

Method	Synthetic dataset AP					Synthetic dataset VUS-PR					TSB-U
	$K=1$	$K=2$	$K=4$	$K=8$	$K=16$	$K=1$	$K=2$	$K=4$	$K=8$	$K=16$	AP
Non-neural baselines											
CART	0.318	0.362	0.404	0.499	0.553	0.358	0.419	0.490	0.563	0.620	0.156
Logistic regression	0.503	0.587	0.641	0.643	0.742	0.598	0.673	0.739	0.746	0.831	0.289
Random Forest	0.479	0.575	0.720	0.807	0.871	0.576	0.668	0.829	0.903	0.941	0.287
Extra Trees	0.531	0.620	0.731	0.813	0.882	0.650	0.729	0.837	0.905	0.940	0.358
HistGB	0.467	0.611	0.774	0.844	0.898	0.555	0.712	0.873	0.920	0.951	0.306
XGBoost	0.519	0.640	0.776	0.847	0.900	0.613	0.749	0.869	0.922	0.952	0.317
Neural networks trained from scratch											
FCN	0.648	0.675	0.783	0.835	0.865	0.699	0.731	0.824	0.869	0.889	0.310
TCN	0.320	0.425	0.565	0.709	0.800	0.379	0.518	0.670	0.801	0.882	0.235
BiLSTM	0.373	0.414	0.600	0.695	0.779	0.444	0.496	0.699	0.791	0.865	0.288
Transformer	0.565	0.669	0.802	0.876	0.908	0.621	0.728	0.851	0.919	0.941	0.322
Time-series foundation models: zero-shot ($K = 0$)											
MOMENT			0.185					0.207			0.240
TSPulse			0.172					0.218			0.311
Time-RCD			0.820					0.871			0.445
Time-series foundation models: fine-tuned											
MOMENT	0.545	0.612	0.751	0.817	0.875	0.664	0.733	0.857	0.897	0.933	0.379
TSPulse	0.310	0.377	0.499	0.566	0.646	0.352	0.424	0.545	0.616	0.698	0.262
Time-RCD	**0.846**	**0.886**	0.906	0.926	**0.943**	**0.899**	**0.935**	**0.951**	**0.963**	**0.974**	0.511
Coding-agent construction											
Standard (Luna)	0.792	0.853	**0.914**	**0.944**	0.940	0.817	0.884	0.940	0.961	0.958	0.544
FAROS (Luna)	0.779	0.817	0.904	0.938	0.935	0.819	0.838	0.925	0.950	0.948	**0.551**
FAROS (Qwen)	0.778	0.784	0.859	0.799	0.791	0.823	0.837	0.900	0.840	0.834	0.549

4.5 Evaluation metrics

Our primary metric is non-interpolated average precision (AP), computed separately for each evaluation series without point adjustment. FAROS averages include only successfully generated and evaluated detectors. For the synthetic dataset, we average AP within each anomaly type and then equally across types. For TSB-AD-U, we follow the original benchmark's equal-series weighting and average AP across evaluation series (Liu & Paparrizos, 2024). For the synthetic dataset, we also report VUS-PR (Paparrizos et al., 2022) as a supplementary metric, with its computation detailed in Appendix C.

5 Results

5.1 RQ1: Detector construction from few labeled events

Few-label capability. Table 1 compares FAROS at $B = 50$ with non-neural supervised classifiers, neural models trained from scratch, and pretrained time-series models in zero-shot and fine-tuned settings. On the synthetic dataset, FAROS with Luna achieves a mean AP of 0.875 across the five labeled-event budgets and outperforms every tested non-neural supervised classifier and neural model trained from scratch at each K. These results show that coding agents can construct strong executable detectors from limited labeled supervision.

Competitiveness across benchmarks. Pretrained time-series models remain strong competitors. Fine-tuned Time-RCD achieves a higher mean AP and VUS-PR than FAROS on the synthetic dataset, while FAROS with Luna reaches a higher AP on TSB-AD-U. Overall, FAROS is competitive with

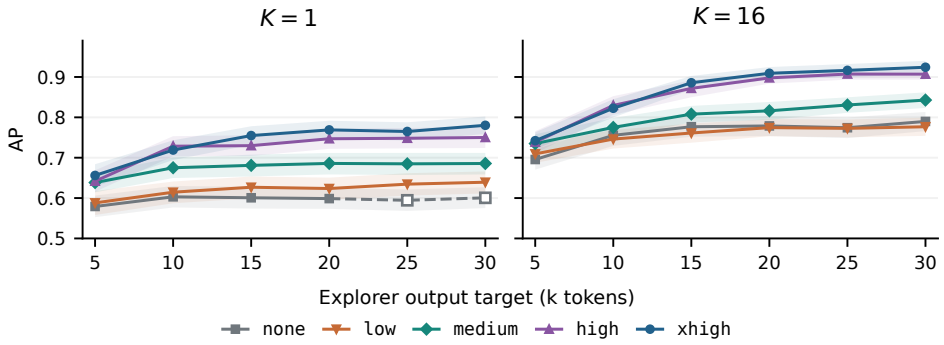


Figure 3: AP across common Explorer-output targets for $K = 1$ and $K = 16$. Shading shows pointwise 95% bootstrap intervals. Open squares in the $K = 1$ panel mark capped-state substitutes for unreached targets in the `none` setting.

pretrained time-series models, although the relative ranking depends on the benchmark and evaluation metric.

Construction across Explorer backbones. The result is not specific to the Luna Explorer. Qwen with `xhigh` reasoning and the common Luna Finalizer achieves a mean AP of 0.802 on the synthetic dataset and nearly matches Luna on TSB-AD-U. Its performance is not monotonic in the labeled-event budget, peaking at $K = 4$ before declining at larger budgets. Additional analysis shows that this decline is also observed on the support data and is concentrated primarily in trend and frequency anomalies (Appendix F). These results suggest that, under the fixed $B = 50$ response budget, additional support examples are not always translated into better detector constructions.

Comparison with a standard agent workflow. A standard coding agent achieves a mean AP of 0.888 across the five label budgets, compared with 0.875 for FAROS at $B = 50$. We find no evidence of a difference in run-to-run variance between the two workflows (Appendix F). Both workflows therefore achieve detector quality of similar magnitude, providing a workflow-level validation for FAROS.

Answer. Coding agents can construct competitive executable anomaly detectors from few labeled events. FAROS with Luna achieves higher AP than the tested non-neural and from-scratch neural baselines at all five labeled-event budgets and remains competitive with specialized pretrained detectors. A standard coding agent also achieves high detector quality, indicating that this construction capability extends beyond the FAROS evaluation workflow.

5.2 RQ2: Reasoning, Search Effort, and Repeatability

Reasoning strength and search length are not interchangeable. Figure 3 compares detector quality at common cumulative Explorer-output targets. From 5k to 30k, the AP gap between `xhigh` and `medium` widens from 0.007 to 0.081 at $K = 16$ and from 0.018 to 0.095 at $K = 1$. Stronger reasoning can also achieve higher quality with fewer output tokens. At $K = 16$, `xhigh` at 10k reaches 0.823 AP, exceeding `low` at 30k with 0.777 AP. Bootstrap comparisons at 30k further distinguish both `high` and `xhigh` from `medium` at both label budgets (Appendix G).

To compare search behavior across reasoning settings, we analyze Python code generated by the Explorer during search, including analysis scripts and candidate detector implementations. We divide each search into successive 5k-output-token intervals and measure code reuse between adjacent intervals. Specifically, for each interval, we compute the fraction of five-token Python sequences that also appeared in the immediately preceding interval. This analysis uses only Explorer-generated code. Figure 4 shows substantially higher reuse for `none` and `low` than for `high` and `xhigh` at both label budgets. At $K = 1$, for example, reuse in the 25–30k interval is 0.771 for `none` and 0.709 for `low`, compared with 0.407 for `high` and 0.318 for `xhigh`. The same separation holds

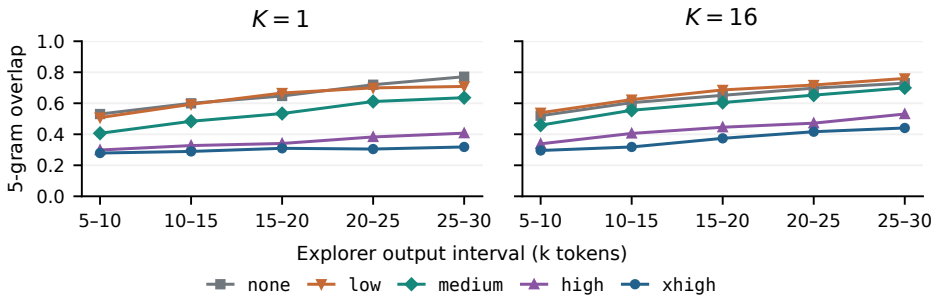


Figure 4: Within-search adjacent-interval lexical 5-gram reuse in Explorer-generated Python code between successive 5k-output-token intervals. Each point compares code generated in that interval with code generated in the immediately preceding interval. The initial 0–5k interval has no preceding interval and is omitted.

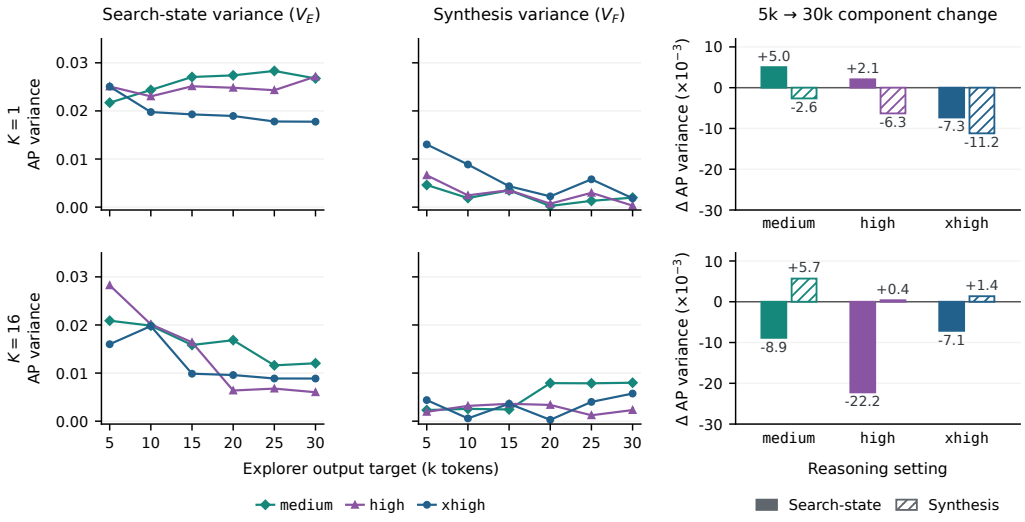


Figure 5: Search-state and synthesis AP variance using the same retained Explorer repetitions across all six output targets. Rows show $K = 1$ and $K = 16$. The left two columns show search-state variance $\widehat{V}_E$ and synthesis variance $\widehat{V}_F$ for `medium`, `high`, and `xhigh`. The right column shows the corresponding changes from 5k to 30k, in units of 10^{-3}. Filled bars denote search-state variance and hatched bars denote synthesis variance. Full trajectories for all five settings, including total variance, are shown in Figure 6 in Appendix H.1.

in fixed-length code comparisons (Appendix H.2). These results show that additional search under lower reasoning is qualitatively different from search under stronger reasoning, with substantially more repeated code across successive stages of exploration. This difference in how additional output is used may contribute to the limited gains from extending lower-reasoning search.

Additional search stabilizes different parts of construction across label budgets. Figure 5 shows that the two sources of variability can change differently as exploration proceeds, and that this pattern depends on label budget. Under `medium` reasoning, search-state variance increases from 5k to 30k at $K = 1$. At $K = 16$, it instead decreases, while synthesis variance appears to increase. Under `high` reasoning, synthesis variance tends to decrease as search proceeds at $K = 1$, whereas search-state variance shows a much larger decrease at $K = 16$. These patterns suggest that additional search can stabilize different parts of the construction process across the two tested label budgets. With limited support, a search can become more reproducible in how its state is converted into a detector while substantial differences across independently reached states remain. With more support,

additional search can instead reduce performance differences across independent searches. Total variance alone does not reveal this distinction.

At the 30k Explorer-output target, search-state variability accounts for 87.8–98.9% of AP variance at $K = 1$ and 60.1–85.1% at $K = 16$ across reasoning settings. It is the larger component in every setting, although synthesis contributes substantial variability in some $K = 16$ settings. Additional analysis shows that this residual synthesis variability is concentrated in a small number of high-disagreement states. Full estimates and diagnostic analyses are reported in Appendix H.1.

Answer. Using FAROS, we find that stronger reasoning and longer search are not interchangeable under the tested budgets. Lower-reasoning searches also show greater code reuse across successive search intervals. Additional search changes search-state and synthesis variability differently, with contrasting patterns across the two tested label budgets. These results show what FAROS adds beyond endpoint-only evaluation. Aggregate endpoint performance does not reveal how repeatability evolves during search, and total run-to-run variability does not identify which stage of construction accounts for those changes. By evaluating matched search states and repeatedly synthesizing from each state, FAROS localizes variability to distinct stages of construction and tracks how their contributions change as search proceeds.

6 Discussion and limitations

Our results suggest that coding agents can construct strong target-adaptive anomaly detectors. The generated detectors are executable numerical programs restricted to standard numerical-computing libraries, yet under some settings they match or exceed the performance of specialized pretrained time-series models. This indicates that substantial detection capability can emerge from the process of analyzing labeled support data, selecting useful detection principles, and assembling them into task-specific code. The standard coding-agent comparison further shows that the detector quality obtained through FAROS is of similar magnitude to that obtained when an agent directly produces its detector at natural termination. This supports the use of state-conditioned synthesis as a practical readout of coding-agent construction states.

RQ2 further shows why evaluating the construction process itself is useful. At matched progress targets, detector quality differs across reasoning settings, showing that the same output budget can correspond to qualitatively different search behavior and detector quality. As search proceeds, search-state and synthesis variance also change differently, and the pattern depends on the tested label budget. This distinction would be obscured by endpoint performance or total run-to-run variance alone. FAROS therefore provides a way to characterize not only how detector quality changes during coding-agent search, but also where variability enters the construction process. More broadly, these results suggest that state-aligned evaluation can be useful for systematically studying coding agents whose search trajectories differ in length, content, and termination behavior across runs.

Several limitations remain. Cumulative Explorer output tokens provide a common progress axis but do not equalize input processing, tool execution, or wall-clock time. The study is limited to target-adaptive time-series anomaly detection, and we evaluate retrospective scoring of anomaly-containing complete series rather than normal-only or streaming settings. The variance analysis uses three to five retained Explorer runs per condition and three synthesis branches per state, so precise component estimates remain subject to finite-sample variation. Finally, state-conditioned performance is a controlled readout through the common Finalizer and should not be interpreted as the exact detector that the continuing Explorer would eventually produce at natural termination.

7 Conclusion

FAROS evaluates coding-agent detector construction at common progress targets and separates search-state from synthesis variability. Our experiments show competitive detector construction from few labeled events. Within the tested output budgets, extending weaker-reasoning search does not recover the quality achieved with stronger reasoning. The two variance components also change differently during search across the tested label budgets. Together, these findings demonstrate how FAROS reveals changes in detector quality and repeatability that endpoint-only evaluation does not resolve.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experiments, assist with translation, and help clean and reformat datasets. We have not used generative AI tools for other tasks requiring disclosure, and the remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to analyze existing literature and edit the manuscript to improve readability. We reviewed all AI-assisted work and verified the resulting experimental procedures, data processing, literature-related claims, and manuscript revisions as appropriate. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This study uses synthetic data and publicly available time-series anomaly-detection benchmarks and does not involve human subjects or private data. Generated detector programs are executed in an isolated environment with restricted filesystem, network, and process access.

REPRODUCIBILITY STATEMENT

The appendices provide the information needed to reproduce the experiments. They document the datasets, prompts, runtime and recorded-state semantics, evaluation and statistical procedures, variance estimators, and baseline configurations. Additional RQ1 and RQ2 analyses and generated-detector examples are also included.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. A review on outlier/anomaly detection in time series data. *ACM Computing Surveys*, 54(3):1–33, 2021.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

Chris U. Carmona, François-Xavier Aubet, Valentin Flunkert, and Jan Gasthaus. Neural contextual anomaly detection for time series. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 2843–2851, 2022.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

Vijay Ekambaram, Subodh Kumar, Arindam Jati, Sumanta Mukherjee, Tomoya Sakai, Pankaj Dayama, Wesley M. Gifford, and Jayant Kalagnanam. TSPulse: Tiny pre-trained models with disentangled representations for rapid time-series analysis. In *International Conference on Learning Representations*, pp. 62090–62133, 2026.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63:3–42, 2006.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. MOMENT: A family of open time-series foundation models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16115–16152. PMLR, 2024.

Yile Gu, Yifan Xiong, Jonathan Mace, Yuting Jiang, Yigong Hu, Baris Kasikci, and Peng Cheng. Argos: Agentic time-series anomaly detection with autonomous rule generation via large language models. *arXiv preprint arXiv:2501.14170*, 2025.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Tian Lan, Hao Duong Le, Jinbo Li, Wenjun He, Meng Wang, Chenghao Liu, and Chen Zhang. Towards foundation models for zero-shot time series anomaly detection: Leveraging synthetic data and relative context discrepancy. In *Forty-third International Conference on Machine Learning*, 2026.

Jun Liu, Chaoyun Zhang, Jiaxu Qian, Minghua Ma, Si Qin, Chetan Bansal, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Large language models can deliver accurate and interpretable time series anomaly detection. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 4623–4634, 2025.

Qinghua Liu and John Paparrizos. The elephant in the room: Towards a reliable time-series anomaly detection benchmark. In *Advances in Neural Information Processing Systems*, volume 37, pp. 108231–108261, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Carl H. Lubba, Sarab S. Sethi, Philip Knaute, Simon R. Schultz, Ben D. Fulcher, and Nick S. Jones. catch22: CAnonical Time-series CHaracteristics selected through highly comparative time-series analysis. *Data Mining and Knowledge Discovery*, 33:1821–1852, 2019.

Itay Nakash, George Kour, and Ateret Anaby Tavor. Efficient agent evaluation via diversity-guided user simulation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 1627–1648. Association for Computational Linguistics, 2026.

Ignacio Oguiza. tsai: A state-of-the-art deep learning library for time series and sequential data. GitHub, 2023. URL `https://github.com/timeseriesAI/tsai`.

Guansong Pang, Chunhua Shen, and Anton van den Hengel. Deep anomaly detection with deviation networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 353–362, 2019.

John Paparrizos, Paul Boniol, Themis Palpanas, Ruey S. Tsay, Aaron Elmore, and Michael J. Franklin. Volume under the surface: A new accuracy evaluation measure for time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(11):2774–2787, 2022.

Junwoo Park, Kyudan Jung, Dohyun Lee, Hyuck Lee, Daehoon Gwak, ChaeHun Park, Jaegul Choo, and Jaewoong Cho. Delving into large language models for effective time-series anomaly detection. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

Priyam Sahoo, Gaurav Mittal, Xiaomin Li, Shengjie Ma, Benjamin Steenhoek, Pingping Lin, and Yu Hu. AgentLens: Revealing the lucky pass problem in SWE-agent evaluation. *arXiv preprint arXiv:2605.12925*, 2026.

Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: A comprehensive evaluation. *Proceedings of the VLDB Endowment*, 15(9):1779–1797, 2022.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Emmanouil Sylligardos, Paul Boniol, John Paparrizos, Panos Trahanias, and Themis Palpanas. Choose wisely: An extensive evaluation of model selection for anomaly detection in time series. *Proceedings of the VLDB Endowment*, 16(11):3418–3432, 2023.

Xiaoyu Tao, Yuchong Wu, Mingyue Cheng, Ze Guo, and Tian Gao. AnomaMind: Agentic time series anomaly detection with tool-augmented reasoning. *arXiv preprint arXiv:2602.13807*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Evan Z Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, William Song, Vaskar Nath, Ziwen Han, Sean M. Hendryx, Summer Yue, and Hugh Zhang. Planning in natural language improves LLM search for code generation. In *International Conference on Learning Representations*, 2025a.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah Goodman. Hypothesis search: Inductive reasoning with language models. In *International Conference on Learning Representations*, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, pp. 65882–65919, 2025b.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1578–1585, 2017.

Haoting Zhang and Shekhar Jain. LLM-assisted logic rule learning: Scaling human expertise for time series anomaly detection. In *1st ICLR Workshop on Time Series in the Age of Large Models*, 2026.

Zihao Zhou and Rose Yu. Can LLMs understand time series anomalies? In *International Conference on Learning Representations*, pp. 1858–1896, 2025.

A Dataset and support construction

A.1 Synthetic dataset

Synthetic dataset generation. We use a fixed revision of the AnomLLM synthetic generator (Zhou & Yu, 2025).[1] For each anomaly type and split, we generate 400 univariate series of length 1,000. We use seed 3407 for the training split and seed 42 for the evaluation split. To ensure reproducible data generation, we modify the range-anomaly generator to preserve the existing NumPy random state when no function-level seed is provided.

Label budgets and support selection. To ensure that each support series contributes exactly one labeled anomaly event, we filter the training split to series containing exactly one event. This yields 1,187 eligible support series across the eight anomaly types (Table 2). An event is a maximal contiguous interval of positive labels. For each anomaly type, we construct three support draws using deterministic, independently seeded sampling. Each draw samples an ordered pool of 16 eligible series without replacement, and budgets $K \in \{1, 2, 4, 8, 16\}$ use nested prefixes of that pool. Because every selected series contains exactly one event, K is also the number of labeled anomaly events in the support set. The same support series and labels are used across Explorer settings, model conditions, and generation repetitions. Different support draws may overlap.

[1] https://github.com/Rose-STL-Lab/AnomLLM/tree/bf7827d

Table 2: Series counts after filtering. The support pool contains training series with exactly one anomaly event. The evaluation set contains all anomaly-containing series, split into single- and multiple-event cases.

Type	support pool	evaluation series	evaluation single	evaluation multiple
range	145	280	148	132
point	147	283	142	141
freq	95	360	95	265
trend	183	170	170	0
flat-trend	183	170	170	0
noisy-point	158	295	173	122
noisy-freq	89	349	90	259
noisy-trend	187	158	158	0

Evaluation population. The evaluation set contains all 2,065 anomaly-containing series, of which 1,146 contain one anomaly event and 919 contain multiple events. Normal-only series are excluded. The reported results therefore measure retrospective anomaly ranking within series that contain anomalies.

A.2 TSB-AD-U

We construct 27 cases from the official TSB-AD-U tuning and evaluation lists (Liu & Paparrizos, 2024), grouping series by source dataset and application category. All 48 tuning series are used as labeled support. Of the 350 official evaluation series, five have no matching support case and two duplicate support series exactly, leaving 343 univariate series evaluated at their original temporal resolution.

B AGENT RUNTIME, PROMPTS, AND RECORDED STATES

Execution environment and models. Explorer and synthesis branches run in Docker containers with Python 3.12.10, OpenHands 1.16.0, and OpenHands SDK 1.21.0. The numerical environment uses NumPy 2.5.2, pandas 3.0.5, SciPy 1.18.0, and scikit-learn 1.9.0. Luna is accessed through an Azure OpenAI API gateway using the request identifier `gpt-5.6-luna`. `Qwen3.8-27B` is hosted on four NVIDIA V100 GPUs, each with 32 GB of GPU memory.

Explorer data and file access. Support CSVs containing values and labels are mounted read-only at `/inputs/signals`. The task instructions and the support scorer at `/inputs/score.py` are also read-only. Through terminal commands, the Explorer can inspect these files, run Python analyses using the installed packages, implement candidate detectors, and obtain support AP from the scorer. It can create and modify files in `/workspace` and use designated temporary directories. The file-editor tool permits reads only under `/inputs` and `/workspace`, and writes only under `/workspace`. Held-out evaluation files are not mounted into the agent containers. Evaluation results from synthesis branches are not returned to the continuing Explorer.

Container and tool isolation. Containers run under an unprivileged user with a read-only root filesystem, dropped Linux capabilities, and privilege escalation disabled. Terminal filesystem access is restricted with Landlock to `/inputs`, `/workspace`, temporary storage, and installed system and library paths. Runtime state and API credentials are outside the Explorer's accessible filesystem and environment. Terminal processes and their children are denied network access using seccomp, while model API communication is handled outside the Explorer's tool access. These controls allow local data analysis and code execution without exposing held-out data, runtime administration, or model communication to the Explorer.

The following task prompts supplement the OpenHands system instructions and tool schemas. We reproduce the FAROS Explorer and Finalizer prompts and the task prompt for the standard coding agent. The prompts do not specify an anomaly type or preferred detector family, and anomaly-type names are removed from filenames.

Explorer task.

```
Labeled time-series CSVs are available under
`/inputs/signals`. Each CSV has numeric `value` and binary `Label`
columns.

Improve the score of an anomaly detector that generalizes to unseen series.
Use the available tools to analyze the labeled Support data and explore as many
detector implementations and parameter choices as useful.

`/inputs/score.py` is available both as a command and as an importable Python
module. For efficient inner loops, use `from score import SupportScorer`, create
one scorer, and call `evaluate(score_fn)`, `evaluate_rule(path)`, or
`evaluate_many(mapping)` repeatedly inside the same Python process. Support is
loaded once. Each call returns macro Support AUPR and prints it to the terminal
transcript; it does not save candidate programs. Use this scorer as the only
source of label-based detector performance; do not calculate such performance
with a separate metric implementation. Diagnostic analysis of values and labels
is allowed.

Do not create `/workspace/working_rule.py`. The controller will
independently generate deployable working rules from frozen copies of your state.
Keep analyzing, implementing, testing, and improving candidates until the
controller stops the run. Do not exploit filenames, absolute row indices, or
exact series length.
```

Finalizer task.

```
# One-call finalizer phase

This branch contains the Explorer state frozen after a specific number of
completed calls. Use only the evidence already present in this conversation and
workspace. Exactly one provider response is available.

In that response, make exactly one `file_editor` tool call that creates
`/workspace/working_rule.py`. Do not read another file, run a
terminal command, test the implementation, call FinishAction, or answer in
prose. The single file must:

- define exactly the public function `score_fn(x)`;
- return one finite 1-D floating score for each time step, with larger meaning
  more anomalous;
- import only numpy, pandas, math, or statistics;
- faithfully implement the detector you would currently choose for deployment.

The controller will save the tool result and stop before any second LLM call.
```

Standard coding agent task.

```
Labeled time-series CSVs are available under `/inputs/signals`.
Each CSV has numeric `value` and binary `Label` columns.

Develop an anomaly detector that generalizes to unseen series.
Use the available tools and `/inputs/score.py` to analyze the data,
implement candidates, and evaluate their support performance.

Save your final detector in `/workspace/working_rule.py`,
defining `score_fn(x)` to return one anomaly score per timestamp.

The final detector may import only `numpy`, `pandas`, `math`, and `statistics`.
Do not use `scipy`, `collections`, or other libraries in the final detector.
```

Synthesis and deployment constraints. The Finalizer receives the recorded conversation and is restricted to a single file-editor call that creates `/workspace/working_rule.py`. It cannot perform further file reads, execute code, or continue exploration. The Evaluator runs the support-valid program separately and supplies only the univariate value array to `score_fn(x)`. The function must return one finite real-valued score per timestamp. Generated deployment programs may import only `numpy`, `pandas`, `math`, and `statistics` and are executed under filesystem, network, process, and randomness restrictions with a 120-second time limit per series. These constraints apply to the generated detector.

Recorded-state semantics. B denotes the number of completed Explorer responses. A recorded state contains the conversation history and workspace files, but not live process memory or shell sessions. Context condensation is disabled, although long tool outputs may be shortened by the runtime. Workspace files are copied into synthesis branches but are not included automatically in the model input. Because the Finalizer cannot read additional files, synthesis relies on evidence already present in the recorded conversation.

Generation settings. Luna (`gpt-5.6-luna`) uses temperature 1.0 and a maximum of 128,000 output tokens per response, including reasoning tokens. Top-p is not explicitly specified. Qwen (`Qwen3.8-27B`) is served with vLLM using checkpoint revision `1d4bf0f2ff6012fd82039f2fa52739d0dd7c60c0`. For Qwen, we do not override the sampling defaults or impose a client-side output-token cap. No sampling seed is fixed for either model. Repeated synthesis uses separate model calls with the same generation configuration.

C Metrics and statistical analysis

C.1 Performance metrics

For a series with binary labels and anomaly scores, we compute scikit-learn's non-interpolated average precision (AP) [2]:

$$\mathrm{AP} = \sum_n (R_n - R_{n-1}) P_n, \tag{4}$$

where P_n and R_n denote precision and recall at successive score thresholds, with $R_0 = 0$. AP is computed separately for each evaluation series. For FAROS on the synthetic dataset, we first average AP over evaluation series within each anomaly type, then over valid synthesis replicas within each state, Explorer runs, support draws, and anomaly types, using equal weights at each step. On TSB-AD-U, we average over valid synthesis replicas and Explorer runs for each evaluation series, then equally over evaluation series.

We additionally report VUS-PR (Paparrizos et al., 2022). Our reported values use the discrete TSB-AD implementation (Liu & Paparrizos, 2024)[3], with `generate_curve`, `version=opt`, and 250 score thresholds. Its computation is

$$\text{VUS-PR}_{\text{TSB-AD}} = \frac{1}{W+1} \sum_{\ell=0}^{W} \sum_{j=1}^{250} (R_{\ell,j} - R_{\ell,j-1}) P_{\ell,j}, \tag{5}$$

where $P_{\ell,j}$ and $R_{\ell,j}$ are the implementation's range-aware precision and recall at buffer length ℓ and threshold index j, with $R_{\ell,0} = 0$. Thresholds are selected at 250 evenly spaced integer indices in the descending score sequence. This implementation uses the current threshold's precision for each recall increment and averages over integer buffer lengths from 0 to W.

For each series x, we set the maximum buffer length W using TSB-AD's `find_length(x)`, which estimates a characteristic period from autocorrelation.[4] This uses only the series values, so all methods share the same W for that series.

C.2 Uncertainty

In RQ2, we report confidence intervals for the AP curves and AP differences between reasoning settings. These intervals quantify uncertainty in the reported means and differences arising from variation across independent Explorer runs.

For each reasoning setting, label budget, anomaly type, and support draw, we first average valid synthesis results within each Explorer state. We then sample five Explorer runs with replacement

[2] `https://scikit-learn.org/1.9/modules/generated/sklearn.metrics.average_precision_score.html`

[3] `https://github.com/TheDatumOrg/TSB-AD/blob/e0975a5/TSB_AD/evaluation/basic_metrics.py`

[4] `https://github.com/TheDatumOrg/TSB-AD/blob/e0975a5/TSB_AD/utils/slidingWindows.py`

within each condition and recompute the overall mean AP. The same sampled runs are used across all output-token targets, and reasoning settings are resampled independently.

For Figure 3, we construct a separate interval for each reasoning setting and output target from the bootstrap distribution of its mean AP. For each reported comparison between two settings, we subtract their mean APs within each bootstrap replicate and construct an interval from the resulting differences. We use 30,000 bootstrap replicates for the RQ2 comparisons at $K = 1$ and $K = 16$. The 2.5th and 97.5th percentiles of the bootstrap estimates define the pointwise 95% confidence intervals.

D NESTED VARIANCE ESTIMATION

Variance decomposition. For the RQ2 analysis, a condition c fixes anomaly type, support draw, label budget, and reasoning setting. We retain an Explorer run only when all three syntheses are valid at all six output targets, and use the same retained runs at every target. Let $\mathcal{R}_c$ denote the retained Explorer runs and $R_c = |\mathcal{R}_c|$, with $R_c \in \{3, 4, 5\}$, and let $J = 3$. For a fixed output target, let u_{rj} denote the mean evaluation AP across the held-out series of the corresponding anomaly type for synthesis j from retained Explorer run r. Define

$$\bar{u}_r = \frac{1}{J}\sum_{j=1}^{J} u_{rj}, \qquad \bar{u} = \frac{1}{R_c}\sum_{r\in\mathcal{R}_c} \bar{u}_r.$$

We estimate

$$\widehat{V}_F = \frac{1}{R_c}\sum_{r\in\mathcal{R}_c} \frac{1}{J-1}\sum_{j=1}^{J}(u_{rj} - \bar{u}_r)^2, \tag{6}$$

$$\widehat{V}_E = \frac{1}{R_c - 1}\sum_{r\in\mathcal{R}_c} (\bar{u}_r - \bar{u})^2 - \frac{\widehat{V}_F}{J}. \tag{7}$$

Here, $\widehat{V}_F$ measures synthesis variability within a recorded state, while $\widehat{V}_E$ measures variability across independently reached search states after subtracting the estimated synthesis contribution to each state mean. Finite samples can yield negative estimates of $\widehat{V}_E$, which are retained when pooling components. Total single-outcome variance is estimated by $\widehat{V}_E + \widehat{V}_F$.

Pooling and variance shares. Variance components are estimated separately for each anomaly type, support draw, label budget, reasoning setting, and output target. For each label budget, reasoning setting, and target, we first average the three support-draw estimates within each anomaly type and then give equal weight to all eight anomaly types. Conditions are not weighted by the number of retained Explorer runs. Let $\overline{V}_E$ and $\overline{V}_F$ denote the pooled components. We define the estimated search-state share of single-detector variance as

$$q_{\text{search}} = \frac{\max(0, \overline{V}_E)}{\max(0, \overline{V}_E) + \overline{V}_F}. \tag{8}$$

Negative pooled search-state variance estimates are retained in variance summaries. For q_{search}, a negative pooled search component is set to zero so that the share remains between 0 and 1. The share is undefined when the denominator is zero.

Validity. The RQ2 variance analysis therefore characterizes performance variability among Explorer trajectories whose synthesis outcomes remain valid across all six evaluated output targets. Execution validity is reported separately using all synthesis outcomes as the denominator. The TSB-AD-U analysis applies the same variance estimators with $R = J = 3$.

E BASELINE IMPLEMENTATION AND SEARCH SPACES

All supervised baselines use the same support and evaluation data as FAROS. Training, configuration selection, and fine-tuning use only the support set.

Table 3: Hyperparameter search spaces for the feature-based classifiers.

Method	Search space
CART	depth D; leaf $[1, 64]_{\log}$; split $[2, 32]$; criterion $\{\text{Gini}, \text{entropy}\}$; features $\{\text{all}, \texttt{sqrt}, 0.5, 0.75\}$; pruning $[0, 0.02]$.
Logistic regression	$C \in [10^{-5}, 10^{4}]_{\log}$.
Random Forest / Extra Trees	trees $\{100, 200, 300, 400, 500\}$; depth D; leaf $[1, 64]_{\log}$; split $[2, 32]$; features $\{\texttt{sqrt}, 0.5, 1.0\}$; criterion $\{\text{Gini}, \text{entropy}\}$.
HistGB	iterations $\{100, 150, \ldots, 500\}$; learning rate $[0.01, 0.2]_{\log}$; leaves $\{7, 15, 31, 63\}$; depth $\{3, 6, 9, \infty\}$; leaf $[1, 64]_{\log}$; L2 $\{0, 10^{-4}, 10^{-3}, 0.01, 0.1, 1, 10\}$.
XGBoost	trees $\{100, 200, 300, 400, 500\}$; learning rate $[0.01, 0.2]_{\log}$; depth $[2, 10]$; child weight $[1, 16]_{\log}$; L2 $[10^{-4}, 20]_{\log}$; L1 $[0, 5]$; row fraction $[0.6, 1]$; feature fraction $[0.5, 1]$.

$D = \{2, 3, 4, 5, 6, 8, 10, 12, 16, 24, 32, \infty\}$, with ∞ denoting no depth limit. Braces denote discrete choices and $[\cdot]_{\log}$ denotes logarithmic sampling. Other ranges are sampled linearly.

E.1 Feature-based non-neural classifiers

We evaluate CART (Breiman et al., 1984), logistic regression, Random Forest (Breiman, 2001), Extra Trees (Geurts et al., 2006), histogram gradient boosting (HistGB), and XGBoost (Chen & Guestrin, 2016). The first five use scikit-learn (Pedregosa et al., 2011), while XGBoost uses its official implementation. Each timestamp is represented by its observed value and catch24 features computed on centered windows of 31, 101, and 301 points, with endpoint-replication padding at series boundaries. Catch24 consists of the 22 catch22 features together with the window mean and standard deviation (Lubba et al., 2019).[5] Nonfinite features are imputed using support-set medians, and logistic regression additionally standardizes features using support statistics.

The scikit-learn classifiers use their balanced class-weighting rule based on support-set class frequencies. For XGBoost, we set `scale_pos_weight` to the ratio of normal to anomalous support timestamps. All support timestamps are retained without resampling or additional sample weighting.

For each method and support condition, we run 100 Optuna trials (Akiba et al., 2019) and select the configuration with the highest mean support AP. The selected model is evaluated unchanged on the evaluation set. Table 3 gives the search spaces.

E.2 Neural networks trained from scratch

We use FCN (Wang et al., 2017), TCN (Bai et al., 2018), BiLSTM (Hochreiter & Schmidhuber, 1997; Schuster & Paliwal, 1997), and Transformer (Vaswani et al., 2017) implementations based on tsai (Oguiza, 2023). Each model receives a centered 301-point raw window with endpoint-replication padding and predicts the center timestamp. Table 4 gives the architectures and readouts.

All models are trained from scratch for 50 epochs with class-weighted cross-entropy based on support-set class frequencies, batch size 32, and Adam with decoupled weight decay 0.01. We use a one-cycle learning-rate schedule with maximum learning rate 3×10^{-4}. No validation split, early stopping, or checkpoint selection is used, and the final epoch is evaluated.

E.3 Zero-shot foundation-model references

We evaluate fixed public zero-shot procedures without support labels. MOMENT-1-small uses reconstruction mean squared error on 512-point windows with stride 256. Time-RCD uses its public `model.zero_shot()` interface with batch size 64 and window size 5,000.[6] TSPulse uses the time-plus-FFT configuration from the official anomaly-detection example in granite-tsfm 0.3.2.[7] It uses aggregation window 64, maximum aggregation, smoothing window 8, least-significant scale

[5] https://github.com/DynamicsAndNeuralSystems/pycatch22

[6] https://huggingface.co/thu-sail-lab/Time-RCD#transformers-api

[7] https://github.com/ibm-granite/granite-tsfm/blob/v0.3.2/notebooks/hfdemo/tspulse_anomaly_detection.ipynb

Table 4: Architectures of the neural baselines trained from scratch.

Model	Core architecture	Readout
FCN	Three convolutional blocks with 128/256/128 channels and kernels 7/5/3, each followed by batch normalization and ReLU.	Center feature
TCN	Eight residual blocks with 25 channels, kernel 7, and dilations 1/2/4/8/16/32/64/128. Each block contains two causal convolutions.	Center feature
BiLSTM	One bidirectional LSTM layer with 100 hidden units per direction.	Concatenated center hidden states
Transformer	One encoder layer with width 64, one attention head, feed-forward width 128, dropout .1, and sinusoidal positional encoding.	Center token

Table 5: Foundation-model fine-tuning settings. All models receive 400 optimizer updates.

Model	Input context	Backbone learning rate	Head learning rate
MOMENT-1-small	512	10^{-5}	10^{-3}
TSPulse	512	10^{-4}	10^{-3}
Time-RCD	1,000	10^{-5}	10^{-5}

0.01, and least-significant score 0.1. These configurations are fixed before evaluation and are not selected using evaluation performance.

E.4 Foundation-model fine-tuning

We fine-tune MOMENT-1-small (Goswami et al., 2024), TSPulse (Ekambaram et al., 2026), and Time-RCD (Lan et al., 2026). For each support condition, we initialize each backbone from its corresponding pretrained weights.

For MOMENT and TSPulse, we attach a newly initialized prediction head consisting of layer normalization followed by a linear layer. Each patch representation produces two logits, corresponding to normal and anomalous classes, for every timestamp within the patch. For Time-RCD, we retain its pretrained anomaly-detection head. Both the backbone and prediction head are fine-tuned. The anomaly-class softmax probability is used as the timestamp anomaly score.

MOMENT and TSPulse use 512-point windows with stride 256, while Time-RCD uses 1,000-point windows with stride 500. Predictions at overlapping timestamps are averaged for all three models. Series shorter than the window length are padded by repeating the endpoint value. Time-RCD normalizes each input window independently. On the synthetic dataset, its 1,000-point window covers the complete series.

All three models are trained using timestamp-wise class-weighted cross-entropy. Within each support condition, the weight for class c is $N/(2N_c)$, where N is the number of support timestamps and N_c is the number belonging to class c. To account for overlapping training windows, each timestamp occurrence is additionally weighted by the inverse of the number of training windows containing that timestamp. The loss weight for each occurrence is the product of its class weight and overlap weight, and the mini-batch loss is normalized by the sum of these weights. Padded positions receive zero loss weight.

All three models use AdamW (Loshchilov & Hutter, 2019), batch size 8, weight decay 0.01, gradient clipping at 1.0, and 400 optimizer updates. The learning rates in Table 5 are fixed across support conditions and remain constant during fine-tuning. No validation split, early stopping, or checkpoint selection is used, and the weights after update 400 are evaluated.

F RQ1 additional results and analyses

F.1 TSB-AD-U evaluation and case-level results

Evaluation protocol. For FAROS, each TSB-AD-U case uses three independent Explorer searches to $B = 50$ and three synthesis repetitions from the endpoint state. Both Explorer backbones use

`xhigh` reasoning, with Luna `xhigh` as the common Finalizer. All 243 synthesis outcomes are valid for each backbone. The 16 baselines use the same support and evaluation series with the procedures in Appendix E. The non-neural classifiers, neural models trained from scratch, and fine-tuned models use three random seeds with the support data held fixed. The primary summary weights the 343 evaluation series equally. Table 1 reports this series-macro AP for all methods.

Case-level results. Table 6 reports support size, evaluation size, and FAROS results using Luna as the Explorer for each case.

Table 6: All 27 TSB-AD-U cases with Luna `xhigh` exploration and synthesis, using three Explorer runs and three syntheses per state. AP averages the nine programs and evaluation series within each case. SD is the sample standard deviation of the three Explorer means, each averaging its three syntheses.

Source	Category	support		evaluation	AP	SD
		Series	Events	Series		
Exathlon	Facility	2	6	28	0.941	0.033
IOPS	WebService	2	21	15	0.300	0.158
LTDB	Medical	1	137	8	0.250	0.086
MGAB	Synthetic	1	10	8	0.258	0.369
MITDB	Medical	1	2	7	0.214	0.048
MSL	Sensor	2	2	7	0.624	0.068
NAB	Facility	1	1	8	0.323	0.134
NAB	Synthetic	1	1	5	0.595	0.233
NAB	Traffic	1	2	5	0.501	0.053
NAB	WebService	2	5	4	0.723	0.024
NEK	WebService	1	4	8	0.797	0.098
OPPORTUNITY	HumanActivity	2	3	27	0.681	0.204
SED	Medical	1	6	2	0.712	0.297
SMAP	Sensor	2	2	17	0.525	0.052
SMD	Facility	5	9	33	0.809	0.081
SVDB	Medical	2	87	18	0.497	0.080
Stock	Finance	2	2,917	8	1.000	0.000
TAO	Environment	1	468	2	0.895	0.155
TODS	Synthetic	2	250	13	0.638	0.034
UCR	Environment	1	1	15	0.158	0.035
UCR	Facility	1	1	15	0.061	0.041
UCR	HumanActivity	1	1	15	0.016	0.010
UCR	Medical	2	2	12	0.222	0.153
UCR	Sensor	1	1	13	0.122	0.064
WSD	WebService	5	16	20	0.751	0.037
YAHOO	Synthetic	2	4	13	1.000	0.000
YAHOO	WebService	3	5	17	0.744	0.012
Total / series-macro AP		48	3,964	343	0.551	—

F.2 QWEN EXPLORATION AT LARGER LABEL BUDGETS

We examine the non-monotonic label-budget effect in the Qwen results of Table 1. All results below use Qwen `xhigh` exploration to $B = 50$ and Luna `xhigh` synthesis. At each K, eight anomaly types, three support draws, three Explorer repetitions, and three synthesis repetitions give 216 outcomes. Support AP is measured by evaluating the synthesized detector on its support set. Both AP summaries use valid detectors and average synthesis repetitions within states, followed by Explorer repetitions, support draws, and anomaly types with equal weights.

Support and evaluation performance. Table 7 shows that the evaluation AP decrease at larger budgets is accompanied by a decrease in support AP. Relative to $K = 4$, support AP decreases by 0.091 at $K = 8$ and 0.083 at $K = 16$, while evaluation AP decreases by 0.060 and 0.067. Thus, the non-monotonic pattern is already visible on the support data. Execution validity remains high across budgets, ranging from 95.8% to 99.5%.

Table 7: Qwen `xhigh` exploration with Luna `xhigh` synthesis at $B = 50$. Support and evaluation AP use the same valid detector outcomes.

K	Support AP	Evaluation AP	Valid outcomes	Validity (%)
1	0.917	0.778	207/216	95.8
2	0.874	0.784	215/216	99.5
4	0.908	0.859	207/216	95.8
8	0.817	0.799	213/216	98.6
16	0.825	0.791	213/216	98.6

Concentration by anomaly type. Table 8 separates the decline by anomaly type. From $K = 4$ to $K = 8$, the largest evaluation AP decreases occur for flat-trend (0.148), freq (0.119), and trend (0.080). These three types account for 72.6% of the net decrease in mean AP. The pattern is not uniform across anomaly types. At $K = 16$, range achieves 0.994 evaluation AP, while noisy-point improves from 0.935 to 0.954. Together, these results show that the non-monotonic label-budget effect is already present on the support data and is concentrated in specific anomaly types. Under the fixed $B = 50$ search horizon, additional labeled examples are therefore not always converted into better detector constructions.

Table 8: Type-level support and evaluation AP for Qwen `xhigh` exploration with Luna `xhigh` code synthesis at $B = 50$.

Anomaly type	Support AP			Evaluation AP		
	$K = 4$	$K = 8$	$K = 16$	$K = 4$	$K = 8$	$K = 16$
flat-trend	0.960	0.775	0.860	0.899	0.751	0.846
freq	0.919	0.779	0.760	0.841	0.722	0.687
noisy-freq	0.680	0.501	0.607	0.573	0.543	0.561
noisy-point	0.989	0.972	0.985	0.935	0.939	0.954
noisy-trend	0.747	0.677	0.649	0.688	0.640	0.603
point	0.999	0.958	0.992	0.977	0.929	0.945
range	1.000	1.000	0.999	0.994	0.985	0.994
trend	0.971	0.877	0.748	0.964	0.884	0.742

F.3 STANDARD-AGENT WORKFLOW SANITY CHECK

We compare FAROS with a standard coding agent that uses Luna with `xhigh` reasoning and directly returns its final detector at natural termination, without a separate Finalizer call. Both workflows use the same support and evaluation series, data-analysis tools, final-program library restrictions, and evaluator. The standard coding agent's task prompt is provided in Appendix B. For each of eight anomaly types, five label budgets, and three support draws, we run three independent standard coding agent searches, giving 360 construction outcomes. The FAROS comparison uses Luna `xhigh` exploration at $B = 50$ and Luna `xhigh` synthesis.

We compare run-to-run variance for a single detector within each anomaly-type, label-budget, and support-draw condition. The primary analysis retains the 117 of 120 conditions in which all standard coding agent and FAROS outcomes are valid. For the standard coding agent, we use the sample variance across its three independent runs. For FAROS, we use the single-outcome variance estimate $\widehat{V}_E + \widehat{V}_F$ from Appendix D. We then compare the condition-level variance difference, defined as standard coding agent minus FAROS. We test the mean variance difference overall and separately at each K using cluster-robust inference, clustering conditions that share an anomaly type and support draw. Table 9 shows that all pointwise 95% confidence intervals include zero, providing no evidence of a variance difference overall or at any individual label budget.

Table 9: Run-to-run variance comparison on conditions with all outcomes valid. Differences are standard coding agent minus FAROS. Intervals are pointwise 95% confidence intervals.

K	Conditions	Standard coding agent	FAROS	Difference	95% interval
1	23	0.01428	0.01623	−0.00195	[−0.01544, 0.01154]
2	24	0.02180	0.01083	0.01097	[−0.00457, 0.02651]
4	23	0.00946	0.00850	0.00095	[−0.00424, 0.00615]
8	23	0.00643	0.00470	0.00173	[−0.00925, 0.01271]
16	24	0.00329	0.01313	−0.00984	[−0.02064, 0.00095]
All	117	0.01107	0.01070	0.00038	[−0.00582, 0.00657]

G RQ2 PERFORMANCE AND RESOURCE ANALYSES

G.1 DETECTION PERFORMANCE ACROSS OUTPUT TARGETS

Table 10 reports AP for all five reasoning settings and six Explorer-output targets at $K = 1$ and $K = 16$. The entries are the means plotted in Figure 3.

Table 10: AP across reasoning settings and Explorer-output targets. Output targets are reported in thousands of tokens. Each panel gives one labeled-event budget. † The mean includes a $B = 1000$ capped-state substitute, as in Figure 3.

$K = 1$						
Setting	5k	10k	15k	20k	25k	30k
`none`	0.579	0.603	0.601	0.599	0.594†	0.601†
`low`	0.588	0.615	0.627	0.624	0.634	0.639
`medium`	0.638	0.675	0.681	0.686	0.685	0.686
`high`	0.642	0.728	0.730	0.747	0.748	0.750
`xhigh`	0.656	0.719	0.755	0.769	0.765	0.780
$K = 16$						
Setting	5k	10k	15k	20k	25k	30k
`none`	0.696	0.755	0.777	0.779	0.774	0.790
`low`	0.709	0.746	0.761	0.775	0.773	0.777
`medium`	0.735	0.775	0.808	0.816	0.831	0.843
`high`	0.739	0.830	0.872	0.898	0.907	0.907
`xhigh`	0.742	0.823	0.886	0.909	0.916	0.924

Output-target states. For the output-target experiments, we select the first completed Explorer response whose cumulative output reaches or exceeds each target, subject to a limit of $B = 1{,}000$ completed responses. All targets are reached except the 25k and 30k targets for one $K = 1$, `none`-reasoning search. In this search, the Explorer stopped invoking tools and repeatedly returned short responses indicating that no further detector work had been performed despite repeated prompts to continue exploration. These responses increased the response count while contributing few output tokens, causing the search to reach the response limit before either target. For both targets, we use the final capped state from that search and explicitly mark aggregate points containing these substitutes in the figures.

G.2 COMPARISONS BETWEEN REASONING SETTINGS

Table 11 reports mean AP differences between reasoning settings at the 30k Explorer-output target. We first average valid synthesis outcomes within each state, then resample five Explorer runs with replacement within each anomaly-type/support-draw condition. Reasoning settings are resampled independently. Pointwise 95% percentile intervals are computed from 30,000 bootstrap replicates of the differences in mean AP.

At both label budgets, the intervals for `high` minus `medium` and `xhigh` minus `medium` exclude zero. The intervals for `xhigh` minus `high` include zero, so the comparison does not resolve a difference between these two settings.

Table 11: AP differences at the 30k Explorer-output target. Each entry gives the mean difference and its pointwise 95% bootstrap interval. Positive values favor the first setting. Intervals are not adjusted for multiple comparisons.

Comparison	$K = 1$	$K = 16$
`low` − `none`	$+0.0387$ $[+0.0004, +0.0756]$	-0.0132 $[-0.0460, +0.0195]$
`medium` − `none`	$+0.0850$ $[+0.0477, +0.1221]$	$+0.0529$ $[+0.0227, +0.0839]$
`high` − `none`	$+0.1495$ $[+0.1128, +0.1865]$	$+0.1171$ $[+0.0900, +0.1443]$
`xhigh` − `none`	$+0.1796$ $[+0.1456, +0.2128]$	$+0.1342$ $[+0.1054, +0.1631]$
`medium` − `low`	$+0.0464$ $[+0.0084, +0.0848]$	$+0.0661$ $[+0.0365, +0.0957]$
`high` − `low`	$+0.1108$ $[+0.0737, +0.1492]$	$+0.1303$ $[+0.1041, +0.1567]$
`xhigh` − `low`	$+0.1410$ $[+0.1066, +0.1762]$	$+0.1474$ $[+0.1196, +0.1753]$
`high` − `medium`	$+0.0644$ $[+0.0272, +0.1024]$	$+0.0642$ $[+0.0411, +0.0876]$
`xhigh` − `medium`	$+0.0946$ $[+0.0606, +0.1296]$	$+0.0813$ $[+0.0560, +0.1067]$
`xhigh` − `high`	$+0.0302$ $[-0.0040, +0.0645]$	$+0.0171$ $[-0.0044, +0.0380]$

Comparisons involving `none` at $K = 1$ include the capped-state substitute shown in Figure 3.

G.3 Resource use

Table 12 reports mean Explorer response counts and cumulative Explorer output tokens at the 30k target, together with support and evaluation AP of the synthesized detectors. At $K = 16$, `low` uses an average of 176.8 responses, compared with 36.7 for `xhigh`, but achieves lower evaluation AP, 0.777 versus 0.924. Thus, similar output-token budgets can correspond to substantially different numbers of interactions, and more interactions do not necessarily yield better detectors.

Table 12: Resource use and detector performance at the 30k Explorer-output target.

K	Effort	Responses (times)	Output (k tokens)	Support AP	Evaluation AP
1	`none`†	188.9	30.0	0.974	0.601
1	`low`	209.2	30.1	0.951	0.639
1	`medium`	134.3	30.1	0.981	0.686
1	`high`	51.7	30.3	0.994	0.750
1	`xhigh`	30.7	30.6	0.992	0.780
16	`none`	153.3	30.1	0.831	0.790
16	`low`	176.8	30.1	0.821	0.777
16	`medium`	123.5	30.2	0.887	0.843
16	`high`	59.1	30.4	0.962	0.907
16	`xhigh`	36.7	30.6	0.959	0.924

† Capped-state substitute. See Figure 3.

H RQ2 repeatability and search-behavior analyses

H.1 Repeatability

Common-output-target results. Figure 6 shows total, search-state, and synthesis AP variance across the six Explorer-output targets at $K = 1$ and $K = 16$. It reports all five reasoning settings, including `none` and `low`, which are not shown in Figure 5. Table 13 reports the component estimates at the 30k output target.

Concentration of synthesis variability. At the 30k output target, each retained Explorer run contributes one recorded state, from which three detectors are synthesized. Using the same retained Explorer runs and weighting as the primary analysis, we rank states by their contributions to pooled synthesis variance separately for each label budget and reasoning setting. For each K and reasoning

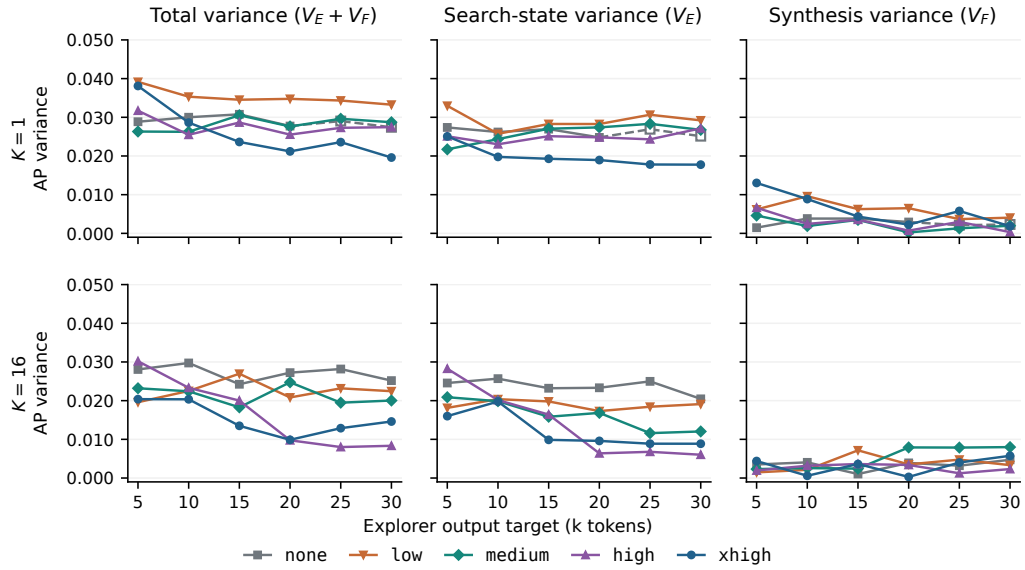


Figure 6: AP variance across all five Luna reasoning settings. Rows show $K = 1$ and $K = 16$. Columns show total variance $\widehat{V}_E + \widehat{V}_F$, search-state variance $\widehat{V}_E$, and synthesis variance $\widehat{V}_F$. The analysis uses the same fixed successful Explorer repetitions as Figure 5. Each condition retains the same three to five Explorers across all six output targets, with three syntheses per state. All 24 anomaly-type/support-draw conditions remain in each setting and receive equal weight. Open squares and dashed segments include capped-state substitutes for `none` at $K = 1$.

Table 13: AP variance at the 30k output target using fixed retained Explorer repetitions. An Explorer repetition is retained only if all 18 synthesis outcomes across the six targets are valid. All 24 anomaly-type/support-draw conditions remain in each setting. R gives the range of retained Explorer counts across conditions, with three syntheses per state. † Includes the capped-state substitute marked in Figure 3.

K	Setting	R	Search-state $\overline{V}_E$	Synthesis $\overline{V}_F$	Total	q_{search} (%)
1	`none`†	5	0.025	0.002	0.027	91.6
1	`low`	4–5	0.029	0.004	0.033	87.8
1	`medium`	4–5	0.027	0.002	0.029	93.1
1	`high`	4–5	0.027	< 0.001	0.027	98.9
1	`xhigh`	4–5	0.018	0.002	0.020	90.6
16	`none`	4–5	0.020	0.005	0.025	81.3
16	`low`	4–5	0.019	0.003	0.022	85.1
16	`medium`	3–5	0.012	0.008	0.020	60.1
16	`high`	4–5	0.006	0.002	0.008	72.2
16	`xhigh`	5	0.009	0.006	0.015	60.7

Totals and shares use unrounded components. The actual R is used in each sample variance.

setting, we analyze 115–120 retained states. Table 14 shows that just three states with the largest contributions account for 89.9–99.9% of synthesis variance at $K = 1$ and 66.3–85.7% at $K = 16$. Synthesis variability is therefore concentrated in a small number of states.

Table 14: Share (%) of pooled synthesis variance contributed by the three highest-contributing states at the 30k output target. The denominator is synthesis variance, not total variance. † The $K = 1$ analysis includes the capped-state substitute marked in Figure 3.

Setting	$K = 1$	$K = 16$
`none`†	95.8	75.3
`low`	99.9	66.3
`medium`	98.2	82.2
`high`	89.9	85.7
`xhigh`	98.2	83.2

For each of `medium` and `xhigh` at $K = 16$, Table 15 reports the three evaluation AP values for the state contributing most to pooled synthesis variance. Although all three detectors from each state are valid, one has substantially lower evaluation AP than the other two.

Table 15: Evaluation AP of three syntheses from each of two high-disagreement states at the 30k output target. Support-seed, Explorer, and synthesis identifiers are zero-based. AP_j denotes the AP of synthesis replica j.

K	Setting	Type	Support seed	Explorer	AP_0	AP_1	AP_2
16	`medium`	point	0	3	0.993	0.038	0.993
16	`xhigh`	trend	2	1	0.938	0.938	0.047

Execution validity. Table 16 reports synthesis validity and the number of Explorer repetitions retained by the fixed-Explorer criterion. Of the 21,600 synthesis outcomes, 21,570 are valid (99.86%). The analysis retains 1,181 of the 1,200 Explorer repetitions while preserving all 240 data conditions and all six output targets.

Table 16: Execution validity and retained Explorer repetitions. Target columns give valid synthesis outcomes out of the original 360 outcomes per setting and target, before any filtering. The final column gives retained Explorer repetitions out of 120 per label-budget and reasoning-setting combination. † Includes the capped-state substitute marked in Figure 3.

K	Setting	5k	10k	15k	20k	25k	30k	Retained Explorers
1	`none`†	360	360	360	360	360	360	120
1	`low`	359	360	359	356	360	360	116
1	`medium`	360	360	359	358	358	360	118
1	`high`	360	360	360	360	359	360	119
1	`xhigh`	360	360	360	359	360	360	119
16	`none`	360	360	360	360	359	358	117
16	`low`	360	360	360	360	358	360	118
16	`medium`	360	359	357	355	360	358	115
16	`high`	360	359	360	360	360	360	119
16	`xhigh`	360	360	360	360	360	360	120

Sensitivity to validity handling. The main analysis uses Fixed Explorers, which removes an Explorer run if any of its three syntheses is invalid at any of the six output targets. This preserves the same Explorer runs across targets while retaining every data condition. We compare this choice with two alternatives. Complete curve retains a data condition only if all 90 outcomes from five Explorer runs, three syntheses per state, and six output targets are valid. Failure-adjusted retains all outcomes

and assigns zero utility to invalid ones. Table 17 compares the resulting search-state variance shares at 30k. The overall attribution is stable across validity treatments. Search-state variability remains the larger component in every setting, although the estimated share is most sensitive for $K = 16$ with `medium`, where it ranges from 50.6% to 64.2%.

Table 17: Search-state variance share q_{search} (%) at 30k under alternative treatments of invalid synthesis outcomes. Fixed Explorers is the analysis used in the main results. † Includes the capped-state substitute marked in Figure 3.

K	Setting	Fixed Explorers	Complete curve	Failure-adjusted
1	`none`†	91.6	91.6	91.6
1	`low`	87.8	82.2	87.0
1	`medium`	93.1	93.2	92.8
1	`high`	98.9	98.3	98.8
1	`xhigh`	90.6	89.5	90.5
16	`none`	81.3	81.3	76.0
16	`low`	85.1	85.6	85.2
16	`medium`	60.1	64.2	50.6
16	`high`	72.2	71.3	72.5
16	`xhigh`	60.7	60.7	60.7

TSB-AD-U results. We additionally apply the variance decomposition to the 27 TSB-AD-U cases at $B = 50$, using Luna `xhigh` for both exploration and synthesis, with three Explorer runs and three syntheses per state. Each detector's outcome is its mean AP across evaluation series within the case. All 243 synthesis outcomes are valid. Under equal-case pooling, search-state variance accounts for 99.4% of the estimated single-detector variance. These estimates suggest that differences across Explorer states contribute more to performance variability than repeated synthesis from a fixed state in this experiment.

H.2 Explorer code reuse during search

We analyze Python code generated during the Explorer searches at $K \in \{1, 16\}$ across five reasoning settings. Each search is divided into successive 5k-output-token intervals up to the 30k target. Python code is extracted from terminal commands and file-editor actions, and workspace and input paths are normalized before lexical tokenization. The analysis includes analysis scripts and candidate detector implementations generated during exploration, but not detectors produced during final synthesis.

For each interval w, we define adjacent-interval reuse as the fraction of lexical 5-gram occurrences in the current interval that also occur in the immediately preceding interval,

$$R_w^{\text{prev}} = \frac{\sum_g C_w(g)\mathbf{1}\{C_{w-1}(g) > 0\}}{\sum_g C_w(g)}, \tag{9}$$

where $C_w(g)$ denotes the count of 5-gram g in interval w. The first interval and comparisons for which either interval lacks extracted 5-grams are undefined. Conditions are weighted equally after averaging eligible Explorer runs within each anomaly-type/support-draw condition.

To control for differences in code quantity, we also compute the same reuse measure using only the first 512 lexical code tokens from each adjacent interval and, separately, only the last 512 tokens. These comparisons require both intervals to contain at least 512 code tokens.

Table 18 reports the full-code and fixed-length comparisons for the 25–30k output interval relative to the preceding 20–25k interval. Adjacent-interval reuse is higher for `none` and `low` than for `high` and `xhigh` at both label budgets. The same broad separation appears in the equal-length comparisons, showing that lower-reasoning searches generate substantially more code that repeats patterns from the immediately preceding stage of exploration.

Table 18: Python-code reuse in the 25–30k output interval relative to the preceding 20–25k interval. R^{prev} uses all extracted code, while R_{head} and R_{tail} use the first and last 512 lexical code tokens from each interval.

K	Setting	R^{prev}	R_{head}	R_{tail}
1	`none`	0.771	0.600	0.623
1	`low`	0.709	0.563	0.562
1	`medium`	0.636	0.480	0.479
1	`high`	0.407	0.256	0.190
1	`xhigh`	0.318	0.219	0.146
16	`none`	0.729	0.589	0.585
16	`low`	0.760	0.646	0.650
16	`medium`	0.699	0.568	0.563
16	`high`	0.531	0.351	0.299
16	`xhigh`	0.441	0.316	0.214

H.3 Code structure and scoring behavior

Comparison protocol. We analyze the 21,600 synthesis outcomes from the six-target Luna experiments at $K \in \{1, 16\}$, of which 21,570 yield valid programs. Within each fixed anomaly type, support draw, K, reasoning setting, and output target, we compare detector pairs synthesized from the same recorded state with pairs synthesized from independent Explorer searches. All valid programs are evaluated on the same 16 held-out series per anomaly type.

Similarity measures. We compare detector behavior using Spearman rank correlation between timestamp anomaly-score rankings, averaged equally across evaluation series for which the correlation is defined. To compare code structure, we use multiset Jaccard similarity over AST patterns after removing formatting, comments, docstrings, source locations, and variable and function names. We also measure top-5% overlap, the fraction of one detector's top-scoring timestamps that also appear in the other detector's top-5% set.

Aggregation across syntheses and searches. For detectors synthesized from the same recorded state, we average pairwise similarities within each state and then equally across states. For detectors from independent searches, we average similarities within each pair of Explorer runs and then equally across run pairs. Both summaries give equal weight to anomaly types and support draws. Invalid detectors and undefined comparisons are excluded.

Results. Table 19 reports results at the 30k output target. Across all 60 combinations of K, reasoning setting, and output target, mean within-state agreement exceeds mean agreement across independent searches for Spearman rank correlation, AST similarity, and top-5% overlap. The same pattern holds for top-1% and top-10% overlap. These results show that repeated synthesis from the same recorded state tends to preserve code structure and scoring behavior, while independently reached search states produce less similar detectors across all three measures.

Table 19: Detector agreement at the 30k Explorer-output target. For each measure, columns compare detector pairs from independent Explorer searches and from the same recorded state. Values average anomaly types and support draws equally. The dagger marks inclusion of a capped-state substitute.

		Spearman ρ		AST similarity		Top-5% overlap	
K	Setting	Independent	Same state	Independent	Same state	Independent	Same state
1	`none`†	0.348	0.975	0.397	0.742	0.513	0.964
1	`low`	0.349	0.971	0.396	0.770	0.541	0.969
1	`medium`	0.349	0.969	0.418	0.826	0.558	0.969
1	`high`	0.480	0.985	0.443	0.868	0.597	0.984
1	`xhigh`	0.451	0.972	0.468	0.845	0.607	0.973
16	`none`	0.497	0.968	0.395	0.729	0.535	0.957
16	`low`	0.444	0.953	0.397	0.751	0.538	0.950
16	`medium`	0.485	0.959	0.420	0.789	0.591	0.961
16	`high`	0.409	0.982	0.442	0.842	0.641	0.972
16	`xhigh`	0.453	0.947	0.475	0.832	0.655	0.967

I EXAMPLES OF GENERATED DETECTORS

We reproduce the complete source of three example detectors generated by Luna `xhigh` exploration and synthesis at $B = 50$ and $K = 1$. Each reported AP is the performance of the individual detector on its anomaly type's evaluation set.

Range anomalies. The detector scores absolute deviation from the series median, normalized by a robust scale estimated from the median absolute deviation with interquartile-range and standard-deviation fallbacks. Its evaluation AP is 0.999.

```
import numpy as np


def score_fn(x):
    values = np.asarray(x, dtype=float)
    magnitude = np.max(np.abs(values))
    if np.isfinite(magnitude) and magnitude > 0.0:
        values = values / magnitude
    center = np.median(values)
    deviation = np.abs(values - center)
    scale = 1.4826 * np.median(deviation)
    if not np.isfinite(scale) or scale <= 0.0:
        iqr = np.percentile(values, 75.0) - np.percentile(values, 25.0)
        scale = iqr / 1.349 if np.isfinite(iqr) and iqr > 0.0 else np.std(values)
    if not np.isfinite(scale) or scale <= 0.0:
        scale = 1.0
    return deviation / scale
```

Frequency anomalies. The detector fits a two-lag linear predictor to the input series in each time direction, iteratively excluding the largest residuals from fitting. It averages the forward and backward absolute prediction residuals and applies a three-point moving average. Its evaluation AP is 0.883.

```
import numpy as np


def score_fn(x):
    x = np.asarray(x, dtype=float)
    n = x.size
    if n <= 2:
        return np.zeros(n, dtype=float)

    def residuals(z):
        m = z.size
        design = np.column_stack((z[1:-1], z[:-2], np.ones(m - 2)))
        target = z[2:]
        coef = np.linalg.lstsq(design, target, rcond=None)[0]
        for _ in range(8):
            residual = np.abs(target - design @ coef)
            keep = residual <= np.quantile(residual, 0.8)
            coef = np.linalg.lstsq(design[keep], target[keep], rcond=None)[0]
        return np.r_[0.0, 0.0, np.abs(target - design @ coef)]

    forward = residuals(x)
    backward = residuals(x[::-1])[::-1]
    raw = (forward + backward) / 2.0
    padded = np.pad(raw, (1, 1), mode="reflect")
    return np.mean(np.lib.stride_tricks.sliding_window_view(padded, 3)[:n], axis=1)
```

Flat-trend anomalies. The detector selects a lag with low robust dispersion of lag differences, scores deviations from the median difference at that lag, and adds each residual to the scores of both timestamps in the pair. Its evaluation AP is 0.909.

```
import numpy as np


def score_fn(x):
    x = np.asarray(x, dtype=float)
    n = x.size
    if n < 6:
        return np.asarray(np.abs(x - np.median(x)), dtype=float)

    lo = 2
    hi = max(lo + 1, min(n - 1, n // 3))
    dispersion = np.empty(hi - lo + 1, dtype=float)
    for offset, lag in enumerate(range(lo, hi + 1)):
        delta = x[lag:] - x[:-lag]
        center = np.median(delta)
        dispersion[offset] = np.median(np.abs(delta - center))

    best = float(np.min(dispersion))
    local_minima = []
    for offset in range(1, dispersion.size - 1):
        if dispersion[offset] <= dispersion[offset - 1] and dispersion[offset] <= dispersion[offset + 1]:
            local_minima.append(offset)
    tolerance = 1.5 * best + 1e-12
    offset = next(
        (candidate for candidate in local_minima if dispersion[candidate] <= tolerance),
        int(np.argmin(dispersion)),
    )
    lag = lo + offset

    delta = x[lag:] - x[:-lag]
    center = np.median(delta)
    residual = np.abs(delta - center)
    score = np.zeros(n, dtype=float)
    score[lag:] += residual
    score[:-lag] += residual
    return score
```