DIAGCODER: TRAINING LANGUAGE MODELS TO DIAGNOSE MULTIVARIATE TIME-SERIES ANOMALIES VIA EXECUTABLE CODE

Anonymous authors
Paper under double-blind review

ABSTRACT

Diagnosing anomalies in multivariate cloud telemetry requires identifying affected metrics and time intervals and explaining their abnormal behavior. Existing approaches often produce entity-level anomaly scores, make one-shot language-model judgments, or rely on predefined diagnostic operators, limiting their ability to investigate context-dependent anomalies. We propose DIAGCODER, an agentic framework that uses executable code as its action space for multivariate time-series diagnosis. Guided by diagnostic perspectives rather than a fixed workflow, DIAGCODER iteratively constructs and executes analyses tailored to temporal and cross-metric context, producing metric-level anomaly intervals with inspectable computational evidence. To learn effective diagnostic strategies, DIAGCODER jointly optimizes analysis construction and diagnostic decision-making through end-to-end reinforcement learning, without requiring expert reasoning trajectories. Built on Qwen3.5-27B and trained only on industrial cloud telemetry, DIAGCODER achieves the best overall performance across the industrial dataset and public benchmarks, outperforming existing anomaly-detection approaches and leading closed-source large language models within the same framework.

1 INTRODUCTION

Cloud platforms monitor servers through correlated metrics spanning computation, memory, storage, and networking. When a performance incident occurs, detecting that a server is anomalous is only the beginning of the investigation (Chen et al., 2024). In multivariate telemetry, operators must determine which metrics are affected, identify the corresponding time intervals, and understand the evidence supporting these judgments. This requires more than recognizing deviations in individual series. A spike may correspond to a recurring workload peak, a persistent level shift may reflect a benign operating change, and a weak deviation may become informative when considered alongside related metrics. Multivariate anomaly diagnosis therefore requires context-dependent analysis that connects metric-level observations to the broader behavior of the system.

Existing methods address different aspects of this task. Deep multivariate detectors learn temporal and inter-metric dependencies (Hundman et al., 2018; Xu et al., 2022; Tuli et al., 2022; Zamanzadeh Darban et al., 2024), but their interpretability is often limited: detection outputs primarily take the form of anomaly scores, which do not directly explain the observed abnormal behavior, the basis for the judgment, or how changes across metrics are related. Post-hoc dimension attribution provides additional clues (Su et al., 2019; Li et al., 2021), but is distinct from an explicit account of the analyses supporting a diagnosis. LLM-based approaches introduce natural-language reasoning and explanations through prompting or task-specific fine-tuning (Alnegheimish et al., 2024; Sun et al., 2026; Kong et al., 2025). However, judgments generated directly from the input lack active computational verification of suspected anomalies. Agentic methods offer a promising direction for interpretable diagnosis by coupling model reasoning with computational evidence. ARGOS (Gu et al., 2025) and AnomaMind (Tao et al., 2026) pursue this direction through rule generation and tool-assisted analysis, respectively. Both focus on univariate time-series analysis, without explicitly addressing joint diagnosis across metrics.

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.

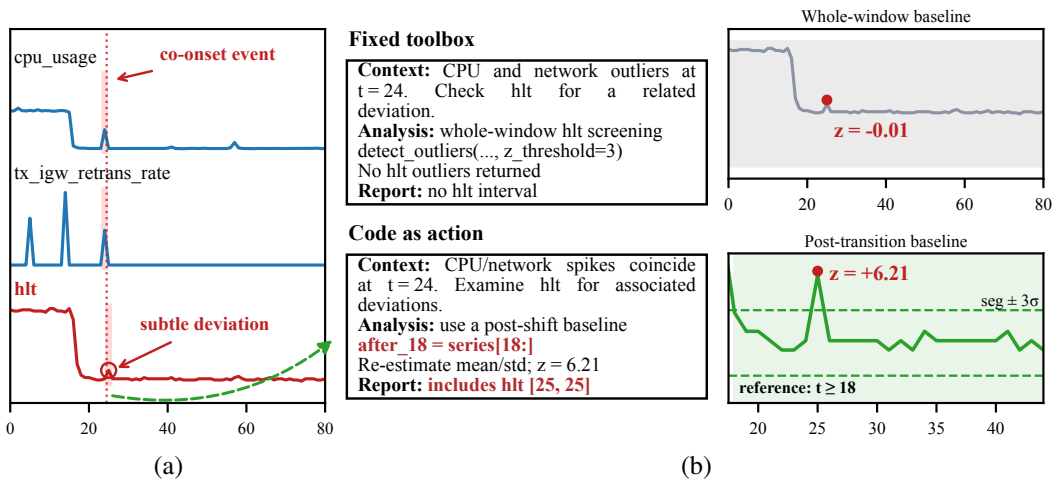


Figure 1: Context-dependent anomaly analysis through executable code. (a) Co-occurring changes near $t \approx 24$ provide cross-metric context for a subtle deviation in `hlt` (three of 40 anomalous metrics shown). (b) Schematic comparison under a shared diagnostic context, using analysis calls and results from the agents' executions. Whole-window screening returns no `hlt` outliers. The code-based agent constructs a post-shift reference and re-estimates statistics on $t \geq 18$, revealing the deviation at $t = 25$ ($z = 6.21$). The right-hand plots contrast the whole-window and post-transition references.

Extending this paradigm to multivariate diagnosis requires analyses tailored to temporal and cross-metric context. A fixed statistical toolbox allows the agent to select operators and adjust parameters, but its predefined interfaces constrain which analyses can be performed directly. Context-dependent diagnosis may require selecting a reference segment, constructing a derived series, or combining computations tailored to a newly identified pattern. Figure 1 illustrates this distinction: a deviation obscured by whole-window statistics becomes apparent against a segment-specific baseline. Executable code allows the agent to construct such analyses during the investigation. However, this flexibility also requires effective strategies for choosing computations and interpreting their results. The challenge is therefore to combine flexible analysis construction with model training, particularly for open-weight models that can be deployed on private telemetry.

We present DIAGCODER, an agentic framework that uses executable code as its action space to investigate multivariate time-series anomalies. We guide this investigation with temporal and cross-metric diagnostic perspectives rather than a prescribed workflow, allowing the model to construct and execute analyses tailored to each input. The model iteratively refines its analyses and diagnosis using execution feedback, producing metric-level anomaly intervals and cross-metric interpretations with an inspectable record of the supporting computations. We further apply end-to-end reinforcement learning (RL) to optimize how the model selects analyses and uses execution results to make diagnostic decisions. The Qwen3.5-27B-based model, trained on VMPD alone, achieves the highest average F1 across VMPD, PSM, and MSL, exceeding both existing detectors and four closed-source LLMs evaluated within the same framework. To examine the contribution of code-based analysis, we compare against a fixed-toolbox agent and find larger average gains from RL, although transfer outcomes vary across held-out datasets. Trajectory analysis further suggests that RL improves how intermediate evidence is translated into final diagnoses, through better retention of true anomaly candidates and rejection of false ones.

Our contributions are as follows:

- We introduce DIAGCODER, an agentic framework for multivariate time-series diagnosis that unifies executable analysis construction and diagnostic decision-making, combining metric-level anomaly localization with inspectable computational evidence.
- We show that RL amplifies the advantage of code-based diagnosis over a fixed toolbox, and provide an execution-level account of this interaction through candidate coverage and final-report selection.

- We demonstrate the effectiveness of task-specialized diagnosis: trained on industrial telemetry alone, a self-hostable 27B model achieves higher average F1 across three benchmarks than existing anomaly detectors and all four evaluated closed-source LLMs.

2 Related Work

Time-series anomaly detection. Deep time-series anomaly detection (TSAD) methods detect deviations through forecasting (Hundman et al., 2018), reconstruction (Su et al., 2019; Tuli et al., 2022), or association discrepancy (Xu et al., 2022). OmniAnomaly and InterFusion further identify anomalous dimensions using reconstruction-based evidence (Su et al., 2019; Li et al., 2021). Although useful for localization, anomaly scores and dimension rankings do not themselves explain the temporal and cross-metric behavior underlying a diagnosis. Our focus is metric-level interval diagnosis supported by explicit computational analyses.

LLMs for time-series anomaly detection. LLM-based TSAD has explored prompting and task-specific adaptation. SigLLM (Alnegheimish et al., 2024) examines direct detection and forecasting-based detection, while LLMAD (Liu et al., 2025) retrieves normal and anomalous reference segments for in-context reasoning. LLM-TSAD (Park et al., 2025) combines statistical decomposition with index-aware prompting to improve temporal understanding and localization. ChatAD (Sun et al., 2026) uses evolved multi-turn instruction data for supervised fine-tuning, followed by preference optimization. These methods primarily adapt input representations, prompts, or model parameters; we study how to construct and revise numerical analyses during diagnosis.

Agentic anomaly detection. ARGOS (Gu et al., 2025) generates and refines executable detection rules offline, then deploys them alongside existing detectors without runtime LLM calls. AnomaMind (Tao et al., 2026) follows a coarse-to-fine workflow supported by knowledge memory and numerical tools. It combines general-purpose LLM reasoning with a detection-specific model trained through supervised fine-tuning and RL. Both focus on individual series rather than explicitly addressing joint diagnosis across metrics. In contrast, DiagCoder constructs analyses for each multivariate test window and trains the model end-to-end across the full diagnostic interaction.

Code-based agents. CodeAct (Wang et al., 2024) establishes executable Python as a unified action space, while SWE-agent (Yang et al., 2024) studies interfaces for navigating, editing, and executing code in software engineering. AD-AGENT (Yang et al., 2025) applies multi-agent code generation to assemble anomaly-detection pipelines from existing libraries. Building on executable interaction, we investigate how code-based analysis and RL jointly improve per-window multivariate diagnosis, rather than general software engineering or detector-pipeline construction.

3 Method

Whether an observed change is anomalous depends on its temporal context and the behavior of related metrics. Computational analyses of this context may support an initial judgment or suggest an alternative explanation, prompting the model to revisit relevant metrics, adjust reference segments, or perform additional checks. We therefore formulate multivariate time-series diagnosis as a sequential decision-making process in which analyses and diagnostic judgments are refined through execution feedback. As illustrated in Figure 2, DiagCoder implements this process with executable code as its action space and uses end-to-end reinforcement learning to jointly optimize analysis construction and diagnostic decision-making.

3.1 Problem Formulation

Let $X \in \mathbb{R}^{T \times F}$ denote a multivariate time-series window with T time steps and F metrics, where $X[t, f]$ is the observation of metric m_f at time t. The ground-truth labels are represented by a binary matrix $Y \in \{0, 1\}^{T \times F}$, where $Y[t, f] = 1$ indicates an anomaly in metric m_f at time t.

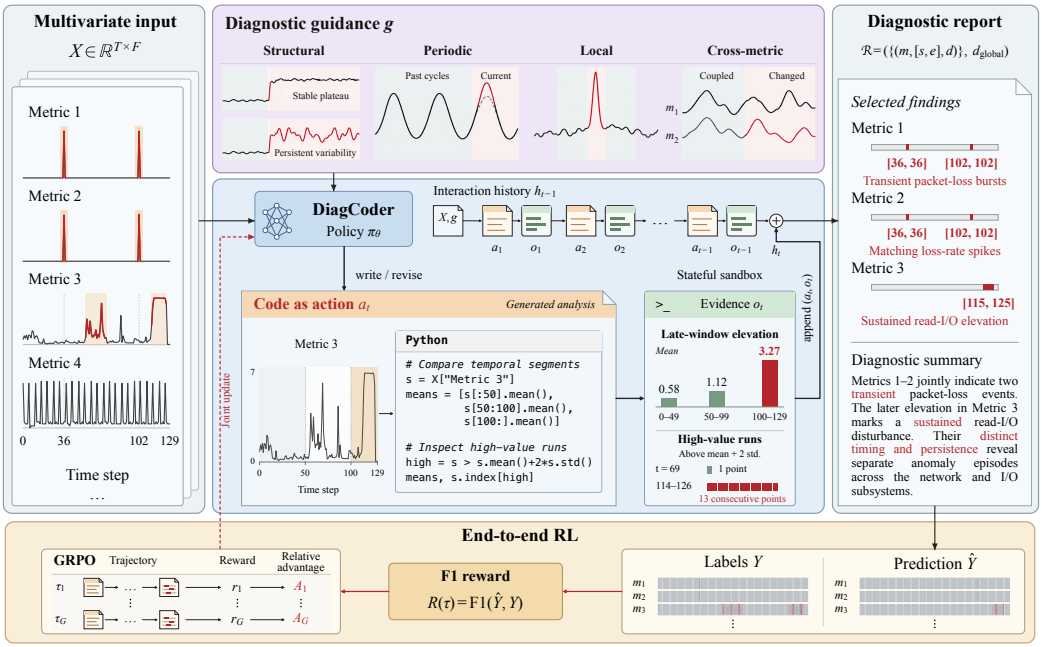


Figure 2: Overview of DIAGCODER. Diagnostic guidance g provides structural, periodic, local, and cross-metric perspectives without prescribing a fixed analysis sequence. Conditioned on the input and interaction history, the policy writes and revises Python programs a_t in a stateful sandbox. Execution evidence o_t informs subsequent analyses and the final report, which contains metric-level intervals, local descriptions, and a diagnostic summary. During training, the reported intervals are converted to a prediction mask $\hat{Y}$ and scored against labels Y using point-level F1. GRPO jointly optimizes analysis construction and diagnostic decisions. Guidance examples are schematic; selected curves, computations, and findings are adapted from VMPD test window 75.

Given X, the task is to localize anomalous intervals in individual metrics and produce an accompanying diagnostic report

$$\mathcal{R} = \left(\{(m_{f_k}, [s_k, e_k], d_k)\}_{k=1}^{K}, d_{\text{global}} \right), \tag{1}$$

where each entry specifies a metric m_{f_k}, an inclusive interval $[s_k, e_k]$, and a local anomaly description d_k. The term d_{global} denotes a report-level diagnostic summary. Here, K is the number of reported intervals, $f_k \in \{1, \ldots, F\}$, and $0 \le s_k \le e_k < T$; multiple entries may refer to the same metric.

3.2 AGENTIC DIAGNOSIS AS SEQUENTIAL DECISION-MAKING

Given a multivariate time-series window X, a language-model policy π_θ investigates potential anomalies by interacting with an execution environment $\mathcal{E}$ through an action interface $\mathcal{A}$. The initial history h_0 contains X and diagnostic guidance g. At each analysis step t, the policy generates an action a_t based on the current history, and its execution yields an observation o_t:

$$a_t \sim \pi_\theta(\cdot \mid h_{t-1}), \qquad o_t = \mathcal{E}(a_t), \qquad h_t = h_{t-1} \oplus (a_t, o_t), \tag{2}$$

where $\mathcal{E}$ maintains execution state across steps and $\oplus$ appends each action–observation pair to the history.

The history h_t serves as the policy's decision context, combining the observed time series with the evidence obtained through analysis. Actions determine how the window is examined, while their results inform subsequent analyses and anomaly judgments. We denote the resulting diagnostic trajectory by τ; the policy aims to produce the report $\mathcal{R}$ from the accumulated history within a turn budget N.

Within this framework, the action interface determines how analyses are expressed, while the policy governs their use in diagnosis. Section 3.3 describes how DIAGCODER implements this process through diagnostic guidance and executable analysis; Section 3.4 optimizes π_θ over complete diagnostic trajectories using their final outcomes. The execution protocol is provided in Appendix A.2.

3.3 CONTEXT-GUIDED EXECUTABLE DIAGNOSIS

DIAGCODER uses diagnostic guidance to frame analytical questions, executable programs to investigate them, and the resulting evidence to form and refine metric-level anomaly judgments.

Diagnostic guidance. The guidance distinguishes identifying a change from judging it anomalous. Spikes, level shifts, trends, and changes in variability provide clues to investigate; their diagnostic meaning is assessed using persistence, surrounding temporal structure, and the behavior of related metrics. Structural, periodic, local, and cross-metric perspectives organize these considerations into questions for investigation. They guide what evidence to seek without prescribing a fixed analysis sequence. The policy selects and combines analyses according to the input and the evidence already available. These principles are supplied through an anomaly-detection skill (Appendix A.1).

Executable analysis. The policy implements its chosen analyses as Python programs that it writes, revises, and executes in a stateful sandbox. Programs can specify both the observations to examine and how to examine them: they can select metrics and time segments, choose reference observations, construct transformed or derived series, and combine statistical computations. These choices translate a diagnostic question into a concrete analysis by defining the observations, comparisons, and quantities used to examine it.

The policy also determines what each program returns for subsequent judgment, such as comparison statistics, time-indexed values, or candidate intervals. For example, Figure 2 pairs segment-level means with high-value runs to examine the magnitude and persistence of an elevation. The required evidence therefore shapes both the computation and its outputs. Programs are constructed to address the current diagnostic question, rather than merely to produce a collection of statistical results.

Execution results are interpreted alongside the input series rather than treated as final anomaly labels. They may support or weaken an initial judgment, or indicate that a different analysis is needed. The policy decides which deviations warrant reporting, determines their metric-specific intervals, and provides explanations grounded in the available evidence. The same policy produces analytical actions and the final report, enabling their joint optimization through the terminal localization reward in Section 3.4.

3.4 JOINT OPTIMIZATION OF EXECUTABLE ANALYSIS AND DIAGNOSIS

The executable interface makes analysis construction a policy decision, but successful execution alone does not establish its localization value. We therefore optimize complete diagnostic trajectories, jointly training analysis construction and diagnostic decisions.

Metric-level labels supervise localization outcomes without prescribing an analysis sequence. The interval component of $\mathcal{R}$ is expanded into a binary prediction mask $\hat{Y} \in \{0,1\}^{T\times F}$: $\hat{Y}[t,f] = 1$ if and only if there exists an entry k with $f_k = f$ and $t \in [s_k, e_k]$; unreported metrics receive all-zero predictions. For the VMPD training distribution $\mathcal{D}$, the terminal reward and expected localization objective are

$$\begin{aligned} R(\tau) &= \mathrm{F1}\big(\hat{Y}(\tau), Y\big), \\ J_{\mathrm{loc}}(\theta) &= \mathbb{E}_{(X,Y)\sim\mathcal{D},\ \tau\sim\pi_\theta(\cdot|X)}\big[R(\tau)\big]. \end{aligned} \tag{3}$$

Point-level F1 supplies task-specific supervision without reference analysis trajectories, preference annotations, or a learned reward model. It scores localization, not intermediate reasoning or diagnostic descriptions.

We use CoBA-RL (Yao et al., 2026) to dynamically allocate a fixed batch-level rollout budget across windows and group-relative policy optimization (GRPO) (Shao et al., 2024) for policy updates. We adapt its capability estimates to continuous rewards, using historical mean trajectory rewards instead of binary success rates. Rewards are normalized within each window's variable-size rollout group

to obtain relative advantages. The policy loss covers all generated tokens, including analytical actions and final reports, but excludes execution outputs. Both components share the terminal reward, without per-computation rewards. Entropy and reference-policy KL regularize the loss separately from the task reward. Appendix A.3 provides the training details.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate on one industrial and two public multivariate benchmarks. **VMPD** is a proprietary dataset containing 85 performance metrics from cloud virtual machines, with point-wise, metric-level anomaly annotations. **PSM** (Abdulaal et al., 2021) contains 25 pooled server metrics from eBay's application-server monitoring infrastructure, with shared point-wise labels for server incidents. **MSL** (Hundman et al., 2018) comprises 27 telemetry sequences from NASA's Mars Science Laboratory rover, each with 55 input features and point-wise anomaly labels for the target telemetry variable. Dataset statistics are provided in Appendix B.1.

Baselines. We compare against twelve recent state-of-the-art methods from three families: (i) *deep anomaly detectors*—KAN-AD (Zhou et al., 2025), CATCH (Wu et al., 2025), CrossAD (Li et al., 2025), and DADA (Shentu et al., 2025); (ii) *time-series foundation models*—TimesFM (Das et al., 2024), Chronos-2 (Ansari et al., 2025), Time-RCD (Lan et al., 2026), and TimeRadar (He et al., 2026); and (iii) *LLM-based methods*—AnomaMind (Tao et al., 2026), ARGOS (Gu et al., 2025), LLMAD (Liu et al., 2025), and LLM-TSAD (Park et al., 2025). KAN-AD, CATCH, and CrossAD are trained on each target dataset's training split; DADA and the foundation models are applied zero-shot. The four LLM-based baselines use GPT-5.5 as their primary backbone to reduce backbone-related variation, while retaining their method-specific auxiliary models and use of training information.

Evaluation protocol. DIAGCODER uses Qwen3.5-27B as its backbone and is trained with RL only on the VMPD training split. The trained policy is evaluated on PSM and MSL without target-dataset adaptation to assess out-of-distribution transfer. We report point-level micro precision, recall, and F1 without point adjustment (Kim et al., 2022). On VMPD, true positives, false positives, and false negatives are counted over metric–time pairs in the $T \times F$ label matrices. On PSM, we OR-aggregate predictions across metrics at each time step and evaluate them against the shared temporal labels. On MSL, we OR-aggregate predictions across retained input features and evaluate them against the temporal anomaly labels of the target telemetry variable.

4.2 MAIN RESULTS

DIAGCODER outperforms all twelve competing methods in both average F1 and precision across the three benchmarks (Table 1), with a 10.7% improvement in average F1 over LLMAD, the strongest prior method. It surpasses all evaluated deep anomaly detectors and time-series foundation models in both average precision and recall. Compared with prior LLM-based methods, DIAGCODER combines higher average precision with competitive average recall, achieving a more favorable overall precision–recall balance.

The largest gain is on VMPD, where DIAGCODER improves F1 over LLMAD by 28.6%, highlighting its strength in metric-level anomaly localization in industrial cloud telemetry. Trained only on VMPD, DIAGCODER also achieves the highest F1 on MSL and remains competitive with prior LLM-based methods on PSM, without target-dataset adaptation.

4.3 PERFORMANCE COMPARISON WITH FRONTIER LLMS

Table 2 compares DIAGCODER with four frontier closed-source LLMs—GPT-5.5, Claude Opus 4.8, Gemini 3.1 Pro, and Qwen3.7-Plus—within the same code-based diagnostic framework. All configurations share the diagnostic guidance, interaction budget, and output protocol (Appendix A).

DIAGCODER achieves the highest average F1 across the three benchmarks, outperforming all four evaluated frontier LLMs. Its average F1 exceeds that of GPT-5.5, the strongest competing configura-

Table 1: Comparison with twelve recent state-of-the-art methods on three benchmarks without point adjustment. **Bold** and underlining indicate the best and second-best F1 scores, respectively.

	VMPD (in-dist.)			PSM (OOD)			MSL (OOD)			Average		
Method	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Deep anomaly detectors												
KAN-AD	.467	.213	.293	.389	.162	.229	.175	.343	.232	.344	.240	.251
CATCH	.084	.067	.074	.540	.027	.052	.376	.179	.242	.333	.091	.123
CrossAD	.205	.172	.187	.301	.275	.287	.272	.271	.271	.260	.239	.249
DADA	.145	.380	.210	.382	.368	.375	.196	.504	.282	.241	.417	.289
Time-series foundation models												
TimesFM	.260	.583	.360	.294	.497	.370	.227	.271	.247	.260	.450	.325
Chronos-2	.233	.555	.328	.294	.500	.370	.227	.261	.242	.251	.439	.313
Time-RCD	.316	.116	.170	.362	.659	.468	.266	.282	.273	.315	.352	.304
TimeRadar	.112	.224	.150	.445	.600	**.511**	.214	.548	.308	.257	.457	.323
Prior LLM-based methods												
AnomaMind	.204	.569	.300	.307	.606	.408	.198	.375	.259	.236	.517	.322
ARGOS	.590	.134	.218	.323	.092	.143	.190	.110	.139	.368	.112	.167
LLMAD	.366	.577	.448	.303	.576	.397	.269	.398	.321	.313	.517	.389
LLM-TSAD	.249	.667	.363	.307	.582	.402	.220	.430	.291	.259	.560	.352
DIAGCODER	.584	.568	**.576**	.365	.419	.390	.244	.483	**.324**	.398	.490	**.430**

Table 2: Comparison with four frontier LLMs under the same code-based diagnostic framework. **Bold** and underlining indicate the best and second-best F1 scores, respectively.

	VMPD			PSM			MSL			Average		
Model	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Claude Opus 4.8	.488	.597	.537	.327	.316	.322	.196	.213	.205	.337	.376	.354
Gemini 3.1 Pro	.310	.602	.409	.328	.535	.407	.254	.434	.321	.297	.524	.379
GPT-5.5	.396	.769	.523	.329	.474	.388	.225	.445	.299	.317	.563	.403
Qwen3.7-Plus	.334	.668	.446	.310	.699	**.430**	.186	.320	.235	.277	.562	.370
DIAGCODER	.584	.568	**.576**	.365	.419	.390	.244	.483	**.324**	.398	.490	**.430**

tion, by 6.6%. It also achieves the highest F1 on VMPD and MSL, demonstrating that its advantage is not confined to the training-domain benchmark. These results show that a task-specialized 27B model can surpass the evaluated frontier LLMs in overall diagnostic performance under the same agent framework.

4.4 EFFECTS OF CODE-BASED ANALYSIS AND RL

To assess the contributions of code-based analysis and RL, we compare against a tool agent that selects and configures eight predefined statistical operators. Both agents use Qwen3.5-27B and share diagnostic principles, inputs, interaction budgets, and output protocols, differing only in their action spaces (Appendix A.1).

The code-based agent achieves higher F1 than the tool-based agent across all three datasets, both before and after RL (Table 3). After training, DIAGCODER also achieves higher average precision and recall, demonstrating a more favorable overall precision–recall balance. RL further amplifies the advantage of code-based analysis, improving average F1 by 21.0% for the code-based agent, compared with 14.3% for the tool-based agent. These results support combining executable analysis with end-to-end RL to learn effective diagnostic strategies. Comparisons using additional backbones are provided in Appendix C.

Table 3: Performance of tool-based and code-based agents before and after RL. **Bold** and underlining indicate the best and second-best F1 scores, respectively.

		VMPD			PSM			MSL			Average		
Agent Design	RL	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Tool-based	×	.360	.390	.374	.310	.568	.401	.183	.226	.202	.284	.395	.326
Tool-based	✓	.585	.439	.501	.310	.435	.362	.201	.346	.254	.365	.406	.372
Code-based	×	.320	.553	.406	.328	.519	**.402**	.197	.372	.258	.282	.481	.355
Code-based (DIAGCODER)	✓	.584	.568	**.576**	.365	.419	.390	.244	.483	**.324**	.398	.490	**.430**

Table 4: Candidate pools and candidate selection before and after RL on VMPD ($|G| = 20{,}850$). **Bold** indicates the higher selection rate within each agent design.

		Tool-based		Code-based	
Aspect	Measure	RL ×	RL ✓	RL ×	RL ✓
Candidate pool	True candidates $\lvert E \cap G \rvert$	6,247	7,099	13,146	10,524
	False candidates $\lvert E \setminus G \rvert$	25,014	21,475	52,023	31,978
Candidate selection	True-candidate retention	.796	**.839**	.766	**.876**
	False-candidate rejection	.786	**.865**	.707	**.786**

4.5 EFFECTS OF RL ON DIAGNOSTIC BEHAVIOR

We analyze how RL changes candidate generation and diagnostic decision-making on the 115 VMPD test windows. Using ground-truth anomaly sets G, we evaluate candidate pools E extracted from execution outputs through three quantities: true-candidate coverage, true-candidate retention in final reports, and false-candidate rejection (Table 4). Candidate extraction is detailed in Appendix D.

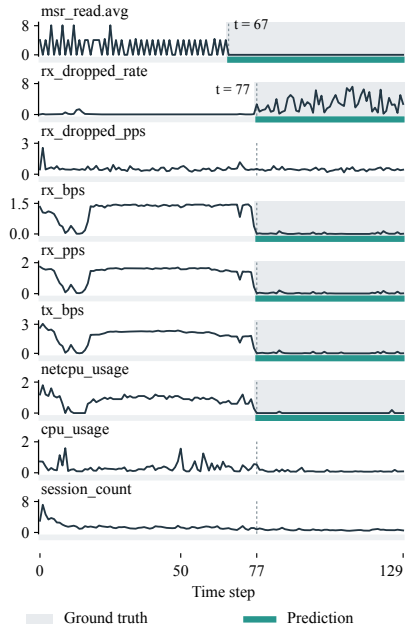
After RL, both agent designs retain a larger fraction of true candidates and reject a larger fraction of false candidates in their final reports. For DiagCoder, RL improves true-candidate retention by 14.4% and false-candidate rejection by 11.1%. These changes accompany substantially higher final precision and slightly higher recall (Table 3), suggesting more effective use of computational evidence.

The code-based agent provides broader true-candidate coverage than the tool-based agent both before and after RL. Its improvement in true-candidate retention also exceeds the 5.4% gain achieved by the tool-based agent. Broader candidate coverage and higher retention after RL are consistent with DiagCoder's higher recall at comparable precision on VMPD.

4.6 CASE STUDY: CONTEXT-GUIDED DIAGNOSIS

Figure 3 illustrates how DIAGCODER uses diagnostic context to construct executable analyses. An increase in receive-side packet drops motivates a targeted investigation of related throughput and network CPU metrics, for which the agent constructs metric-specific reference baselines to identify sustained decreases. For `netcpu_usage`, the final diagnosis identifies an interval matching the labeled anomaly span, even though all 53 anomalous points remain within the whole-window 3σ band.

DIAGCODER also integrates metric-level findings into an event-level diagnosis with inspectable computational evidence. In this example, the final report associates increased packet drops with concurrent decreases in throughput and network CPU activity, describing a network dysfunction event while retaining metric-specific anomaly intervals. The accompanying programs and execution outputs expose the reference segments, statistics, and intervals underlying these findings (Figure 3(b)).



(a)

Executions 02-04: initial analysis

Several metrics show **structural changes**; locate their change points.
rx_dropped_rate **rises near t = 77**, while RX/TX metrics decline.
Examine this metric together with **other network-related metrics.**

Executed check: $\mu + 2\sigma = 4.97$; 11 points flagged.
[89, 98, 105, 109, 110, 111, 114, 115, 117, 121, 128]

Execution 05: network-metric investigation

Inspect network-metric declines after t = 75 to determine whether a **systemic network anomaly** is present.
Selected: rx_bps, rx_igw_bps, ..., tx_bps, netcpu_usage
Author note: rx_bps was absent from the initial candidate list.

```
baseline_mean = series[:50].mean()
baseline_std = series[:50].std()
threshold = baseline_mean - 2 * baseline_std
```

Execution return
rx_bps: reference mean 1.09; lower bound 0.10
[9,9], [13,15], [77,84], [86,121], [123,129]
netcpu_usage: reference mean 0.87; lower bound 0.05
[9,9], [12,16], [77,84], [86,124], [126,129]

Execution 06 and final report

Segment means, t = 0-76 to t = 77-129:
RX drops: 0.09 to 2.86; RX throughput: 1.18 to 0.02.
TX throughput: 1.86 to 0.04; network CPU: 0.85 to 0.01.
Final report: **network dysfunction** from t = 76-77; MSR stops at 67.
rx_bps / tx_bps / netcpu_usage: **[77,129]**; MSR: **[67,129]**

(b)

Figure 3: Context-guided diagnosis on VMPD test window 61. (a) Nine selected metrics in their input scales, with ground-truth anomaly spans in grey and final predicted intervals in green. (b) Abridged diagnostic trajectory with recorded code and execution outputs; red text highlights key observations and decisions.

5 Conclusion

Multivariate time-series diagnosis requires identifying affected metrics and intervals while interpreting their behavior in temporal and cross-metric context. We presented DiagCoder, which addresses this challenge by jointly learning executable analysis construction and diagnostic decision-making through end-to-end RL. This design enables context-dependent investigation with inspectable computational evidence. Trained only on industrial telemetry, our 27B model achieves state-of-the-art average F1 across three benchmarks, outperforming all twelve competing anomaly detection methods. It also surpasses all four closed-source LLMs evaluated within the same agent framework. Further analysis shows that RL amplifies the advantage of code-based diagnosis and improves the selection of intermediate findings for final reporting. DiagCoder establishes an effective, self-hostable approach to multivariate anomaly diagnosis that combines adaptive analysis, state-of-the-art performance, and inspectable evidence.

Limitations and future work. The datasets used in this study support anomaly localization evaluation but lack the annotations needed for systematic assessment of diagnostic reports. We plan to construct expert-annotated diagnostic datasets and establish evaluation protocols for diagnostic correctness, consistency with computational evidence, and cross-metric interpretation. In addition, DiagCoder currently operates on individual time windows. Future work will extend the framework to streaming telemetry by maintaining context across windows and tracking the evolution of anomaly events.

REPRODUCIBILITY STATEMENT

Section 3 describes the diagnostic framework and reinforcement learning objective. Appendix A documents the diagnostic guidance, interaction and output protocols, and training configuration. Section 4.1 and Appendix B describe the datasets, preprocessing, evaluation protocol, and baseline settings. The source code, including training and evaluation scripts and experimental configurations, will be made publicly available upon acceptance. VMPD is planned for release separately as part of a dedicated dataset publication.

AI USE STATEMENT

Generative AI tools assisted with manuscript drafting, language refinement and translation, figure design and production, literature organization, and the interpretation and presentation of experimental results. GPT-5.5 additionally assisted the diagnostic-behavior analysis by extracting explicitly asserted anomaly indices from code-based execution outputs that were not resolved by deterministic rules, subject to source-line and index-range checks (Appendix D.2). All AI-assisted materials were reviewed and revised by the authors, and candidate references were checked against the original sources. The authors take full responsibility for the accuracy, integrity, and final content of the manuscript and its accompanying artifacts.

REFERENCES

Ahmed Abdulaal, Zhuanghua Liu, and Tomer Lancewicki. Practical approach to asynchronous multivariate time series anomaly detection and localization. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2485–2494, 2021.

Sarah Alnegheimish, Linh Nguyen, Laure Berti-Equille, and Kalyan Veeramachaneni. Can large language models be anomaly detectors for time series? In *IEEE 11th International Conference on Data Science and Advanced Analytics*, pp. 1–10, 2024.

Abdul Fatir Ansari, Oleksandr Shchur, Jaris Küken, Andreas Auer, Boran Han, Pedro Mercado, Syama Sundar Rangapuram, Huibin Shen, Lorenzo Stella, Xiyuan Zhang, Mononito Goswami, Shubham Kapoor, Danielle C. Maddix, Pablo Guerron, Tony Hu, Junming Yin, Nick Erickson, Prateek Mutalik Desai, Hao Wang, Huzefa Rangwala, George Karypis, Yuyang Wang, and Michael Bohlke-Schneider. Chronos-2: From univariate to universal forecasting. *arXiv preprint arXiv:2510.15821*, 2025.

Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 674–688, 2024.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 10148–10167, 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.

Hui He, Hezhe Qiao, Yutong Chen, Kun Yi, and Guansong Pang. TimeRadar: A domain-rotatable foundation model for time series anomaly detection. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1602–1613, 2026.

Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 387–395, 2018.

Siwon Kim, Kukjin Choi, Hyun-Soo Choi, Byunghan Lee, and Sungroh Yoon. Towards a rigorous evaluation of time-series anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7194–7201, 2022.

Yaxuan Kong, Yiyuan Yang, Yoontae Hwang, Wenjie Du, Stefan Zohren, Zhangyang Wang, Ming Jin, and Qingsong Wen. Time-MQA: Time series multi-task question answering with context enhancement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 29736–29753, 2025.

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 *International Conference on Machine Learning*, 2026.

Beibu Li, Qichao Shentu, Yang Shu, Hui Zhang, Ming Li, Ning Jin, Bin Yang, and Chenjuan Guo. CrossAD: Time series anomaly detection with cross-scale associations and cross-window modeling. In *Advances in Neural Information Processing Systems*, 2025.

Zhihan Li, Youjian Zhao, Jiaqi Han, Ya Su, Rui Jiao, Xidao Wen, and Dan Pei. Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 3220–3230, 2021.

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*, pp. 4623–4634, 2025.

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 *Advances in Neural Information Processing Systems*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Qichao Shentu, Beibu Li, Kai Zhao, Yang Shu, Zhongwen Rao, Lujia Pan, Bin Yang, and Chenjuan Guo. Towards a general time series anomaly detector with adaptive bottlenecks and dual adversarial decoders. In *International Conference on Learning Representations*, 2025.

Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2828–2837, 2019.

Hui Sun, Chang Xu, Haonan Xie, Hao Li, Yuhao Huang, Chuheng Zhang, Ming Jin, Xiaoguang Liu, Gang Wang, and Jiang Bian. ChatAD: Reasoning-enhanced time-series anomaly detection with multi-turn instruction evolution. *arXiv preprint arXiv:2601.13546*, 2026.

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.

Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. TranAD: Deep transformer networks for anomaly detection in multivariate time series data. *Proceedings of the VLDB Endowment*, 15(6): 1201–1214, 2022.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 50208–50232, 2024.

Xingjian Wu, Xiangfei Qiu, Zhengyu Li, Yihang Wang, Jilin Hu, Chenjuan Guo, Hui Xiong, and Bin Yang. CATCH: Channel-aware multivariate time series anomaly detection via frequency patching. In *International Conference on Learning Representations*, 2025.

Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. In *International Conference on Learning Representations*, 2022.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

Tiankai Yang, Junjun Liu, Michael Siu, Jiahang Wang, Zhuangzhuang Qian, Chanjuan Song, Cheng Cheng, Xiyang Hu, and Yue Zhao. AD-AGENT: A multi-agent framework for end-to-end anomaly detection. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 191–205, 2025.

Zhiyuan Yao, Yi-Kai Zhang, Yuxin Chen, Yueqing Sun, Zishan Xu, Yu Yang, Tianhao Hu, Qi Gu, Hui Su, and Xunliang Cai. CoBA-RL: Capability-oriented budget allocation for reinforcement learning in LLMs. *arXiv preprint arXiv:2602.03048*, 2026.

Zahra Zamanzadeh Darban, Geoffrey I. Webb, Shirui Pan, Charu C. Aggarwal, and Mahsa Salehi. Deep learning for time series anomaly detection: A survey. *ACM Computing Surveys*, 57(1): 15:1–15:42, 2024.

Quan Zhou, Changhua Pei, Fei Sun, Jing Han, Zhengwei Gao, Haiming Zhang, Gaogang Xie, Dan Pei, and Jianhui Li. KAN-AD: Time series anomaly detection with Kolmogorov–Arnold networks. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 79136–79149, 2025.

A Agent Implementation and Training

A.1 Diagnostic Guidance and Action Spaces

The anomaly-detection skill guides both what evidence to seek and how to judge whether an observed change is anomalous. The box below summarizes the diagnostic instructions shared by both agents, emphasizing the judgment criteria and analytical perspectives. Repeated explanations, trigger examples, and duplicate output templates are omitted; the report format is specified in Appendix A.2.

Diagnostic Skill

Anomaly judgment. Distinguish what has changed from whether the change indicates abnormal operation. Spikes, drops, level shifts, trend changes, increased oscillation, and altered correlations provide *pattern evidence* for identifying candidates. Establishing an anomaly requires *diagnostic evidence*, such as persistent instability, impaired operation, departures from reasonable behavior, or support from related metrics and domain semantics. The relevant evidence varies with the anomaly: local shape, structural behavior, and cross-metric relationships need not contribute equally. Assess these observations using duration, stability, and temporal context. Domain heuristics may support interpretation but must not replace analysis of the observed series.

Analytical perspectives. Develop an initial understanding of metric semantics, data dimensions, missing values, and broad patterns, then choose the questions most relevant to the suspected anomaly. The following perspectives may be combined, omitted, or revisited as execution results warrant; they have no required order.

Structural changes. Examine sustained trends, persistent mean shifts, and increases in variability. Determine whether an apparent outlier is part of a broader structural change. For a stable post-change plateau, assess whether it reflects a workload transition or a new normal, or a sustained abnormal state following an event.

Periodicity and temporal context. Check whether recurring or seasonal patterns explain the fluctuations and whether the deviation exceeds the variation expected within the cycle. Distinguish short-term fluctuations from longer-term behavior before classifying a recurring pattern as anomalous.

Local deviations. Locate peaks, valleys, spikes, jumps, and intervals that depart from surrounding behavior or the overall distribution. Determine whether they are isolated events or part of a larger structural change; seek additional contextual evidence when a single method is inconclusive.

Cross-metric relationships. Examine changes in established relationships, coordinated behavior, and possible propagation patterns. Investigate system-level abnormalities even when individual metrics change only weakly. Integrate related findings into a joint interpretation while retaining the affected metrics and their anomaly intervals.

Analysis decisions and feedback. Explain the question each computation addresses and why the chosen method is appropriate. Interpret which judgments its outputs support or weaken, and use remaining uncertainties to select further analyses. Keep the final diagnosis consistent with this evidence.

Coverage and stopping. Seek all metrics with clear anomaly evidence and extend the investigation to unexamined metrics with related semantics, the same metric family or processing chain, or synchronous changes. Stop once conclusions are sufficiently supported and important suspicious metrics have been examined; explain why the available evidence is sufficient.

Code-based execution. The code-based skill asks the agent to implement its chosen analyses in Python. The agent writes and runs scripts using `write_file` and `execute`, and can revise them with `edit_file`. Scripts print key numerical results to standard output, using structured JSON for complex results. The agent interprets these outputs to decide what to compute next. Execution takes place in an isolated sandbox, with the raw input window mounted as JSON.

Tool-based execution. The tool-based skill asks the agent to specify the analysis objective and scope, then select operators, metrics, and exposed parameters (Table 5). After interpreting the returned results, it may adjust parameters, choose another analysis, or conclude. Both agents receive an identical initial user message and share task rewards, report formats, and interaction budgets.

Table 5: Predefined operators available to the tool-based agent.

Diagnostic purpose	Operators
Statistical overview	`compute_feature_statistics`
Structural changes	`trend_analysis`, `changepoint_detection`, `volatility_analysis`
Periodic structure	`seasonality_analysis`
Local deviations	`local_features`, `detect_outliers`
Cross-metric dependencies	`multivariate_relationships`

A.2 Interaction and Output Protocols

Each action in Section 3.2 corresponds to one assistant response, which may request multiple interface operations. Their outputs are returned before the next response. Both agents produce an overall diagnosis followed by metric-level anomaly intervals and descriptions. The natural-language account preceding the `<answer>` block corresponds to d_{global} in Section 3. The structured component follows the textual format below; it is not enforced by an API response schema. Indices are zero-based and interval endpoints are inclusive. Disjoint intervals are listed separately, and an empty array denotes no detected anomalies.

```
<answer>
<Anomalies>[{"metric_name":"metric_A","anomalies":[
  {"interval":[120,133],"description":"brief explanation"}
]}]</Anomalies>
</answer>
```

A.3 Training Configuration

Both agents are trained with CoBA-RL and GRPO policy updates. CoBA-RL allocates a fixed batch-level rollout budget across windows using capability estimates from a sliding-window mean of continuous trajectory rewards. Thus, $n = 8$ specifies the average rollout budget per window; individual group sizes vary. Advantages are normalized within each window's allocated group. KL regularization is applied as a loss term using a low-variance estimator. Table 6 summarizes the training configuration. Code execution uses isolated sandbox containers provisioned on a separate service for each trajectory and released after the rollout.

Table 6: Training configuration for the Qwen3.5-27B agents.

Setting	Value
Training data / epochs	459 VMPD windows / 5
Batch size / rollout budget	32 windows / 256 trajectories
PPO mini-batch size / learning rate	16 / 10^{-6}
KL coefficient / entropy coefficient	0.02 / 0.001
Training / testing decoding	Temperature 1.0 / greedy
Gradient checkpointing	Enabled
Hardware	4 nodes $\times$ 8 NVIDIA H20 (141 GB)

B Experimental Setup and Reproduction

B.1 Datasets and Preprocessing

Table 7: Full-dataset statistics. Series denotes the number of sequences; Dim denotes the input dimensionality before feature filtering, and total length sums time steps across sequences. Anomaly percentages count labeled anomalies over retained metric–time pairs for VMPD and over all time steps in the full PSM and MSL datasets, before evaluation windowing.

Dataset	Domain	Series	Dim	Total length	Anomaly (%)
VMPD	Cloud VMs	574	85	75,423	3.29
PSM	Servers	1	25	220,322	11.07
MSL	Spacecraft	27	55	132,046	5.88

VMPD. VMPD contains monitoring data from 574 cloud virtual machines, covering 85 CPU, memory, I/O, and networking metrics sampled at one-minute resolution. Each device contributes one 130–151-step window with point-wise, metric-level anomaly annotations. The dataset comes with a predefined device-disjoint split of 459 training and 115 test windows, which we use unchanged. VMPD is not publicly available at submission; an anonymized release is planned after the ongoing publication process.

PSM. PSM (Abdulaal et al., 2021) contains 25 pooled server metrics from eBay's application-server monitoring infrastructure. It comprises one multivariate time series, with shared point-wise labels indicating server incidents.

MSL. MSL (Hundman et al., 2018) comprises 27 sequences from NASA's Mars Science Laboratory rover, each containing one target telemetry variable and 54 binary command features. Point-wise labels identify anomalous intervals in the telemetry target.

Preprocessing. For PSM and MSL, we evaluate all 140-step test windows with stride 140, adding an overlapping end-aligned window when needed for full coverage. Constant or entirely missing features are removed from each window.

B.2 Baseline Reproduction

We reproduce all baselines from their official implementations. Hyperparameters follow the original recommendations where available. For VMPD, we additionally search exposed configurations and report the best-scoring setting where oracle selection is used. Our agents use a single model unchanged across the three benchmarks.

Deep anomaly detectors. KAN-AD (Zhou et al., 2025), CATCH (Wu et al., 2025), and CrossAD (Li et al., 2025) are trained on each target dataset's own training split. KAN-AD uses a window length of 32, which performed best; CATCH uses its released thresholding procedure. DADA (Shentu et al., 2025) uses the released pretrained model without target-dataset training.

Time-series foundation models. TimesFM (Das et al., 2024) and Chronos-2 (Ansari et al., 2025) use their official anomaly-detection demonstrations, converting forecast deviations into anomaly predictions. Time-RCD (Lan et al., 2026) uses its released pretrained model, and TimeRadar (He et al., 2026) uses the recommended SPOT guard. All four are evaluated zero-shot.

LLM-based methods. All four use GPT-5.5 as the primary backbone, which achieves the highest average F1 among the closed-source code-based configurations in Table 2. AnomaMind (Tao et al., 2026) additionally uses its released 8B detector. ARGOS (Gu et al., 2025) uses the single-rule variant without ensemble aggregation, with detection rules constructed from training data. LLMAD (Liu et al., 2025) retrieves labeled training windows for in-context detection. LLM-TSAD (Park et al., 2025) uses its statistical decomposition and index-aware prompting over serialized values and plots.

C Additional Experimental Results

Table 8 extends the comparisons in Section 4.4 to all five backbones. Each backbone is evaluated within the same tool-based and code-based frameworks specified in Appendix A.1.

Table 8: Complete results for tool-based and code-based agents across backbones and benchmarks. **Bold** and underlining mark the best and second-best F1 scores.

		VMPD			PSM			MSL			Average		
Backbone	RL	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Tool-based agent													
Claude Opus 4.8	×	.533	.475	.502	.333	.497	.399	.160	.197	.177	.342	.390	.359
Gemini 3.1 Pro	×	.266	.483	.343	.311	.602	.410	.189	.482	.271	.255	.522	.342
GPT-5.5	×	.310	.788	.445	.318	.748	**.446**	.202	.482	.285	.277	.673	.392
Qwen3.7-Plus	×	.271	.572	.368	.302	.712	.424	.207	.429	.279	.260	.571	.357
Qwen3.5-27B	×	.360	.390	.374	.310	.568	.401	.183	.226	.202	.284	.395	.326
Qwen3.5-27B	✓	.585	.439	.501	.310	.435	.362	.201	.346	.254	.365	.406	.372
Code-based agent													
Claude Opus 4.8	×	.488	.597	.537	.327	.316	.322	.196	.213	.205	.337	.376	.354
Gemini 3.1 Pro	×	.310	.602	.409	.328	.535	.407	.254	.434	.321	.297	.524	.379
GPT-5.5	×	.396	.769	.523	.329	.474	.388	.225	.445	.299	.317	.563	.403
Qwen3.7-Plus	×	.334	.668	.446	.310	.699	.430	.186	.320	.235	.277	.562	.370
Qwen3.5-27B	×	.320	.553	.406	.328	.519	.402	.197	.372	.258	.282	.481	.355
DiagCoder	✓	.584	.568	**.576**	.365	.419	.390	.244	.483	**.324**	.398	.490	**.430**

D Diagnostic Behavior Analysis

D.1 Candidate Pools and Selection Measures

To distinguish candidate generation from final diagnostic selection (Section 4.5), we reconstruct a cumulative candidate pool E_i from each window's execution outputs and compare it with the ground-truth anomaly set G_i and final prediction set R_i. All three sets contain metric–time-step pairs; R_i expands reported intervals to their constituent points. The union defining E_i includes all candidates asserted during the trajectory, including those retracted later, so their exclusion from R_i is reflected in the selection measures. Ground-truth labels are used to evaluate the extracted candidates, not to identify assertions in execution outputs.

Table 4 pools counts across the 115 VMPD test windows and all retained metrics, including metrics without labeled anomalies. It reports true-candidate coverage as a count, $\sum_i |E_i \cap G_i|$, and the false-candidate count, $\sum_i |E_i \setminus G_i|$. The selection rates are

$$\text{True-candidate retention} = \frac{\sum_i |R_i \cap E_i \cap G_i|}{\sum_i |E_i \cap G_i|}, \tag{4}$$

$$\text{False-candidate rejection} = \frac{\sum_i |(E_i \setminus G_i) \setminus R_i|}{\sum_i |E_i \setminus G_i|}. \tag{5}$$

These are ratios of pooled counts, rather than averages of per-window rates. The pooled ground-truth set contains $20{,}850$ points.

The selection rates condition on membership in E_i and are distinct from final precision and recall. In particular, R_i need not be a subset of E_i: a reported interval can include points between explicitly asserted indices, and the agent also observes raw series and statistics that do not enter the candidate pool.

D.2 Extraction from Execution Outputs

Both agent designs and training conditions use the same candidate definition: metric-attributed anomaly indices or intervals explicitly asserted in execution outputs. Extraction retains in-window indices, expands intervals to points, and unions candidates by metric across steps. Counts, thresholds, and statistics without explicit anomaly locations are excluded.

Format-specific rules first parse structured operator fields and textual assertions, including index lists, interval fields, index-prefixed values, and enumerations. In unstructured outputs, assertion lines are identified by anomaly, outlier, spike, or change-point terms. NumPy integer wrappers are normalized, and two-element lists denote discrete indices unless an interval label or range delimiter specifies an inclusive span. Metrics are attributed from the enclosing structured field or, for textual assertions, from the assertion line or nearest preceding single-metric context in the same output block. Truncated structured returns are parsed from surviving index fields using the nearest preceding metric key.

GPT-5.5 resolves free-form assertion lines left unmatched by the rules. Decisions are cached, and accepted indices must occur literally on the source line and fall within the evaluation window.

As a consistency check, final prediction masks reconstructed for this analysis reproduce the independently scored true-positive, false-positive, and false-negative counts for all 460 trajectories (two action spaces $\times$ two training conditions $\times$ 115 windows).