LLMDP-U: Compiling LLM Reasoning into Persistent Time-Series Anomaly Detectors

Ziqi Zhu[1] **Guanghui Wang**[1] **Zhihao Lin**[1] **Hao Huang**[1] **Bing Zhu**[2] **Ziyuan Li**[2]
Yingli Liu[2] **Zhendong Bei**[2] **Peiyang He**[1*]

[1]Amazon Web Services [2]HSBC Holdings Plc.

Abstract

Large language models (LLMs) can adapt anomaly detection to heterogeneous time series, but recent LLM-based detectors keep the model in the scoring loop, coupling score production to repeated context construction and model inference. We introduce LLMDP-U, which uses an LLM as a detector compiler: from a statistical profile of a reference prefix and a numerical interface, it generates and validates a series-specific Python program that emits multiple continuous anomaly scores. The retained program then executes without further LLM calls, and a fixed label-free rank fusion produces the final score. Evaluated offline and transductively on TSB-AD-U Eval-350, Opus-compiled programs achieve 0.5498 Average VUS-PR, a $+0.0500$ gain over a no-LLM Canonical Program and the third-highest result on the leaderboard. Performance increases from Haiku to Sonnet to Opus under a shared protocol, while a matched-size exhaustive primitive bank does not reproduce the Opus gain. An audit of 1,194 programs reveals recurring profile-conditioned activity, persistence, and value-support mechanisms. Together, these results establish detector programming as a third design point between fixed numerical detectors and recurring foundation-model inference: adaptation is compiled into a persistent, inspectable artifact, while score execution remains numerical.

1 Introduction

Time-series anomaly detection (TSAD) requires different detector biases across series, yet selecting those biases without anomaly labels remains difficult. A sudden level shift, a broken seasonal pattern, and a novel subsequence call for different evidence. This diversity has produced a collection of statistical, geometric, forecasting, and reconstruction-based detectors, with no single bias dominating every series [14].

Existing TSAD systems either fix detector biases in advance or learn transferable representations through time-series pretraining and target-stage adaptation [6, 7, 12, 14]. This leaves an important design space between fixed detector pools and learned models: constructing series-specific detection logic without anomaly labels or per-series model training.

Large language models (LLMs) offer flexible, series-specific adaptation, but recent LLM-based detectors place the model inside anomaly inference. These systems present values, windows, or plots to a language or vision-language model and ask it to identify anomalous regions [1, 13, 32]. This couples score production to repeated context construction and model invocation. We ask whether series-specific adaptation can instead be compiled once and retained as executable detection logic.

We introduce **LLM D**etector **P**rogramming for **U**nivariate Time Series (**LLMDP-U**), which moves this adaptation into compilation. For each series, the compiler receives a statistical profile of a

*Corresponding author (peiyan@amazon.com).

40th Conference on Neural Information Processing Systems (NeurIPS 2026). Workshop: TS-LIMITS @ NeurIPS 2026.

reference prefix and an interface to a predefined numerical library, then emits a validated Python program. The program may invoke library operators or synthesize additional continuous anomaly scores with the scientific Python stack. Its output is a continuous detector bank $\mathbf{D} \in \mathbb{R}^{T \times M}$.

A fixed label-free fusion stage maps this bank to the final anomaly score through tie-aware ranks, PostShare weighting, and global top-$\lceil \sqrt{M} \rceil$ mean aggregation. LLMDP-U performs per-series adaptation at compilation time, leaves behind a persistent and inspectable numerical detector bank, and then exits the scoring path.

We evaluate LLMDP-U on the univariate track of TSB-AD (TSB-AD-U) under its official protocol. On Eval-350, Opus-compiled programs improve Average VUS-PR by $+0.0500$ over the no-LLM Canonical Program and achieve 0.5498, ranking third on the leaderboard without training or fine-tuning a time-series model. Under a shared development protocol, end-to-end performance rises from Haiku to Sonnet to Opus, while instantiating all supplied primitives at a comparable bank size does not reproduce the Opus gain. A runtime provenance audit of 41,735 emitted columns further shows that Opus produces a broad numerical repertoire: 34.4% are compiler-written detectors rather than direct library calls, and the audit identifies recurring profile-conditioned activity-rate, persistence, and value-support mechanisms within this subset. Together, these results establish detector programming as persistent per-series adaptation: the LLM performs adaptation at compilation time, leaves behind an executable detector system, and exits the scoring path.

2 Related Work

Time-series anomaly detection. Statistical and geometric methods such as Sub-PCA, Matrix Profile, and KShapeAD remain competitive alongside neural approaches including OmniAnomaly, USAD, and Anomaly Transformer [3, 14, 16, 21, 24, 27]. TSB-AD evaluates these methods under a unified protocol with VUS-PR as its primary metric [14].

Pretrained time-series models. Chronos tokenizes real-valued sequences for zero-shot forecasting, while ChronosAD adapts its representations for supervised anomaly detection [2, 11]. Time-RCD, TSPulse, and CHARM target transferable zero-shot or fine-tuned detection [6, 7, 12].

LLMs for anomaly detection. SigLLM, LLMAD, AnomLLM, and related work ask an LLM to return anomaly labels or intervals from values or windows [1, 5, 13, 32]. Later systems add de-seasonalization, index-aware prompts, visual representations, learned reasoning objectives, or specialized numerical agents [9, 10, 15, 17, 22, 25, 29, 31]. These methods retain a language or vision-language model in anomaly inference; LLMDP-U moves adaptation before score execution.

Program synthesis for detection. Code models synthesize executable functions and can revise programs from execution feedback [4, 26]. Selective ensembles such as SELECT and LSCP operate over predefined detector pools, whereas LLMDP-U compiles the series-specific detectors themselves [19, 30]. Program synthesis offers a path from language-model reasoning to executable anomaly-detection logic: ARGOS generates deployable Python rules through supervised feedback, while LLM-assisted rule learning distills labeled judgments into deterministic logic [8, 28]. LLMDP-U turns program synthesis from rule generation into application-time label-free detector construction: for each series, it compiles a reference-prefix profile into multiple continuous scoring functions and leaves final aggregation to one fixed numerical rule.

3 Method

3.1 Problem Setting and Overview

Let $\mathbf{x} = (x_1, \ldots, x_T)$ be a univariate time series and let t_0 denote the boundary of the reference prefix provided by the benchmark. At per-series compilation and score-execution time, the compiler, retained program, and fusion rule receive no target-series anomaly labels. The goal is to produce a continuous anomaly score $\mathbf{s} \in \mathbb{R}^T$ in two stages:

1. **Compilation.** A statistical profile of $\mathbf{x}_{1:t_0}$ is converted into a prompt. The LLM then writes a Python detector program for the series.

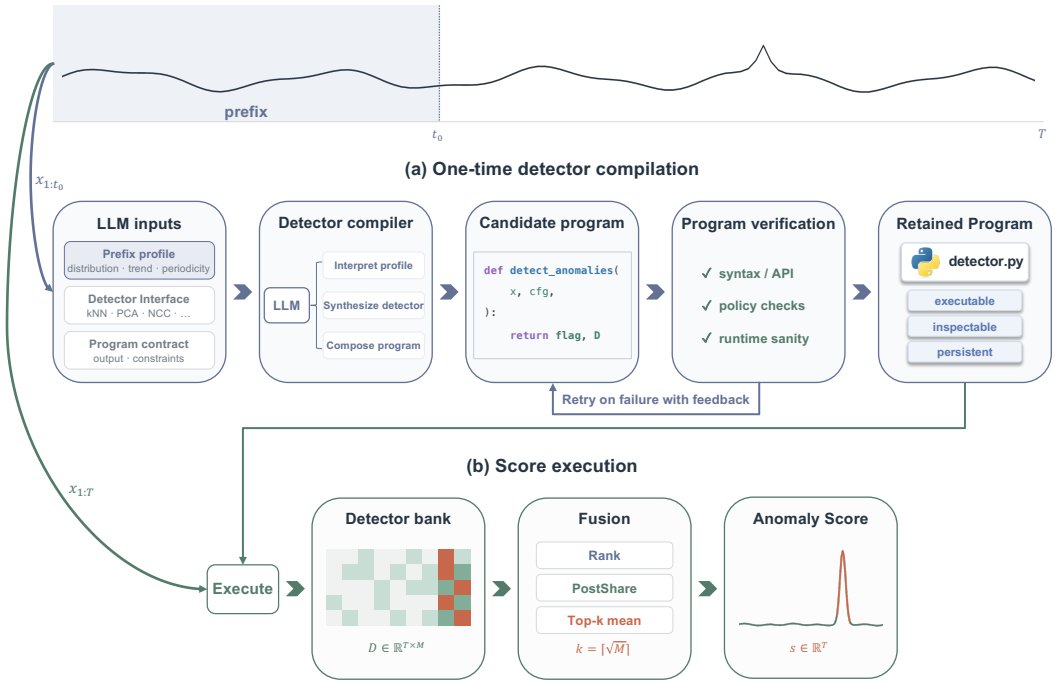


Figure 1: LLMDP-U compiles a detector from the prefix and executes it on the complete series. The prefix is summarized into a statistical profile that, together with the detector contract and `shape_lib`, guides an LLM to produce a validated Python detector. The retained program and complete unlabeled series produce a detector bank $\mathbf{D}$. Tie-aware rank normalization, PostShare weighting, and global top-$\lceil\sqrt{M}\rceil$ pooling yield anomaly score $\mathbf{s}$.

2. **Score execution.** The generated program runs over $\mathbf{x}$ and returns a bank of continuous detector responses. A fixed label-free fusion rule converts this bank into $\mathbf{s}$.

The LLM synthesizes series-specific logic; the scientific Python stack executes it. The compiler sees only the reference-prefix profile, whereas the retained program and fusion stage may use the complete unlabeled series. LLMDP-U is therefore offline and transductive throughout score execution, rather than causal or streaming. Figure 1 summarizes the separation between one-time detector compilation and subsequent numerical score execution.

3.2 Prefix Profiling

The compiler summarizes the benchmark-provided prefix in a fixed-schema JSON profile containing marginal statistics, robust-STL trend and seasonal descriptors, FFT periods, autocorrelation lags, change-point segments, and multi-scale candidate windows. The LLM receives this profile rather than the raw samples; the complete series is accessed only when the retained program is executed.

3.3 Detector-Program Synthesis

The LLM receives the profile, the detector interface, and a code-generation contract. It writes

```
detect_anomalies(series, config) -> (sanity_flag, detector_bank)
```

where `detector_bank` is a finite $T \times M$ matrix

$$\mathbf{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_M] \in \mathbb{R}^{T\times M}.$$

Each column is a named continuous anomaly score. The program contract requires a finite bank of separate continuous scores, forbids internal thresholding or voting, and leaves final aggregation

to the fixed downstream fusion rule. Each column expresses a distinct anomaly hypothesis while preserving its continuous response. This separation lets the compiler construct heterogeneous evidence while one fixed numerical rule aggregates it consistently across series; the retained bank also exposes every component score for inspection.

The fixed `shape_lib` interface exposes six core subsequence primitives: matrix-profile discord, dictionary k-nearest-neighbor distance, template normalized cross-correlation, subsequence PCA reconstruction, cycle residual, and shape changepoint. It standardizes block-wise window extraction, reference-prefix calibration where applicable, projection of window scores back to timestamps, and length-aware execution. The compiler may invoke these primitives or synthesize additional continuous scores using the allowed scientific Python stack.

Before retention, validation checks the function signature, import allowlist, forbidden calls, output shape, and numerical validity. A lightweight runtime test executes the program on spiked and constant series, using the auxiliary sanity flag to reject non-responsive or over-triggering outputs. Failed generations receive validation feedback and may be regenerated. The flag is discarded after validation; benchmark scoring uses only the continuous detector bank. Once retained, the source is stored as the series-specific detector artifact. Score execution runs this fixed program on the complete series to obtain $\mathbf{D}$, without calling the LLM.

3.4 Label-Free Detector-Bank Fusion

Generated programs emit variable-size detector banks whose columns differ in scale and tail behavior. LLMDP-U maps every bank to one anomaly score using the fixed, label-free fusion rule.

Tie-aware ranks. Each detector column is independently converted to percentile ranks, removing scale differences while preserving anomaly order. Equal responses receive the same average rank:

$$r_{t,j} = \frac{\mathrm{rank}_{\mathrm{avg}}(D_{t,j}) - 1}{T - 1}.$$

This produces $\mathbf{R} \in [0,1]^{T \times M}$.

PostShare weighting. PostShare favors detectors whose above-reference response is concentrated after the prefix boundary. For detector j, let

$$\hat{z}_{t,j} = \frac{D_{t,j} - \mu_j}{\sigma_j + \epsilon}, \qquad \tau_j = Q_{0.99}(\hat{z}_{1:t_0,j}),$$

where μ_j and σ_j are the median and scaled MAD of $D_{1:t_0,j}$, respectively. We then define

$$e_{t,j} = \max(\hat{z}_{t,j} - \tau_j, 0), \qquad q_j = \frac{\sum_{t>t_0} e_{t,j}}{\sum_{t=1}^{T} e_{t,j} + \epsilon}, \qquad w_j = 1 + \alpha q_j.$$

Here, w_j is the PostShare weight for detector j, and α controls its strength. The weights are estimated from $\mathbf{D}$ and applied to the rank matrix $\mathbf{R}$. PostShare reads no anomaly labels, although its use of the complete score sequence makes the fusion transductive.

Aggregation. At each timestamp, LLMDP-U averages the $k = \lceil \sqrt{M} \rceil$ strongest weighted ranks:

$$s_t = \frac{1}{k} \sum_{j \in \mathrm{TopK}(\{w_j r_{t,j}\}_{j=1}^{M}, k)} w_j r_{t,j}, \qquad k = \lceil \sqrt{M} \rceil.$$

The same rule is applied to every series. The output $\mathbf{s}$ remains continuous and requires no task-specific decision threshold.

Table 1: Performance on TSB-AD-U Eval-350. Baseline values are unweighted means over Eval-350 in the leaderboard snapshot [23]; LLMDP-U is scored offline on the complete unlabeled series (Section 3.1). TS-pretrained denotes time-series pretraining; "–" denotes an unreported metric.

Method	VUS-PR	VUS-ROC	TS-pretrained
Time-RCD+MAFT (FT)	0.5856	0.9195	✓
TSPulse (FT)	0.5516	–	✓
LLMDP-U (Opus 5)	0.5498 ± 0.0023	0.9183 ± 0.0026	×
Time-RCD	0.5224	0.8711	✓
CHARM	0.5011	–	✓
TSPulse (ZS)	0.4801	–	✓
StreamVAE	0.4508	–	×
MMPAD	0.4399	0.7873	×
Sub-PCA	0.4234	–	×
KShapeAD	0.4008	–	×
xLSTMAD	0.3983	0.8291	×

4 Experiments

4.1 Benchmark and Evaluation

Benchmark. We evaluate on the univariate track of TSB-AD (TSB-AD-U) [14], whose primary metric is Average VUS-PR. The official leaderboard result-file snapshot used for comparison was collected on September 3, 2026 [23]. TSB-AD-U provides a 48-series tuning split for method development and a 350-series evaluation split for final evaluation. Eval-350 spans all 23 source datasets and nine application domains, including industrial monitoring, web services, and healthcare. Each series includes a complete univariate signal, a prefix boundary, and point-level anomaly labels. LLMDP-U receives only the signal and prefix boundary.

Evaluation. We follow the official TSB-AD evaluation protocol. For each series, the tolerance window is estimated using the benchmark's window-selection procedure, and VUS-PR and VUS-ROC are computed with its reference implementation.

Unless otherwise stated, values shown as mean $\pm$ SD denote the mean and sample standard deviation across three compilation runs. 95% percentile bootstrap confidence intervals (CIs) for mean paired differences were estimated using 10,000 nonparametric bootstrap resamples at the series level. For conditions evaluated in three compilations, per-series scores were averaged across compilations before paired differences and CIs were calculated.

4.2 Main Results

LLMDP-U achieves 0.5498 ± 0.0023 Average VUS-PR on Eval-350, ranking third on the collected leaderboard. The two higher VUS-PR entries use pretrained time-series models with target-stage fine-tuning, whereas LLMDP-U trains or fine-tunes no time-series model. The primary configuration uses Claude Opus 5; its synthesis prompt, aggregation rule, and PostShare $\alpha = 0.50$ are selected exclusively on Tuning-48 and frozen for Eval-350. Compiler configuration and validation details appear in Appendix A.1, per-dataset results in Appendix A.7, reliability and runtime results in Appendix A.8, and the complete fusion-selection grid in Appendix A.9. Table 1 compares LLMDP-U with leading methods reported in this TSB-AD-U leaderboard snapshot [23].

4.3 Controlled Compiler Comparisons

The Opus advantage is preserved under a shared final-fusion coefficient and is not reproduced by a matched-size exhaustive primitive bank. Table 2 reports this shared-fusion control alongside the comparison using each configuration's Tuning-48-selected PostShare coefficient. All LLM configurations use the same profiler, synthesis prompt, program contract, aggregation family, evaluator, Tuning-48 split, and candidate PostShare grid. The deterministic controls retain the same reference-prefix profile, profiler-derived temporal scales, numerical stack, fusion family, and evaluator, but replace LLM synthesis. The Canonical Program uses a deterministic recipe combining fixed robust point and residual scores with core `shape_lib` primitives at every eligible profiler-derived scale.

Table 2: Controlled compiler comparisons on Eval-350. Panel A uses Tuning-48-selected PostShare coefficients; Panel B applies the primary configuration's $\alpha = 0.50$ to every detector bank. Within each panel, Δ and 95% CIs are paired against the Canonical Program.

Experiment	α	Bank size	VUS-PR ↑	Δ	95% CI
Panel A: Tuning-48-selected PostShare coefficients					
Canonical Program	0.00	21.52	0.4998	–	–
Exhaustive `shape_lib`	0.25	35.13	0.5016	$+0.0018$	$[-0.0144, +0.0178]$
LLMDP-U (Haiku 4.5)	0.00	23.19	0.4881 ± 0.0030	-0.0117	$[-0.0180, -0.0061]$
LLMDP-U (Sonnet 5)	0.25	22.38	0.5130 ± 0.0071	$+0.0132$	$[-0.0003, +0.0265]$
LLMDP-U (Opus 5)	**0.50**	**34.90**	$\mathbf{0.5498 \pm 0.0023}$	$\mathbf{+0.0500}$	$\mathbf{[+0.0344, +0.0656]}$
Panel B: Shared final-fusion coefficient, $\alpha = 0.50$					
Canonical Program	0.50	21.52	0.5046	–	–
Exhaustive `shape_lib`	0.50	35.13	0.5011	-0.0035	$[-0.0172, +0.0101]$
LLMDP-U (Haiku 4.5)	0.50	23.19	0.4959 ± 0.0034	-0.0087	$[-0.0148, -0.0031]$
LLMDP-U (Sonnet 5)	0.50	22.38	0.5115 ± 0.0048	$+0.0069$	$[-0.0039, +0.0178]$
LLMDP-U (Opus 5)	**0.50**	**34.90**	$\mathbf{0.5498 \pm 0.0023}$	$\mathbf{+0.0452}$	$\mathbf{[+0.0321, +0.0589]}$

Table 3: Compiler-stack ablation on Tuning-48. Each Δ compares the current experiment with the preceding compiler stage.

Experiment	VUS-PR ↑	Δ	95% CI
Raw Prefix	0.3835 ± 0.0229	–	–
Profile Only	0.3894 ± 0.0118	$+0.0059$	$[-0.0572, +0.0717]$
LLMDP-U	$\mathbf{0.5646 \pm 0.0144}$	$\mathbf{+0.1752}$	$\mathbf{[+0.0787, +0.2753]}$

The Exhaustive `shape_lib` control mechanically instantiates all six supplied primitives at every eligible scale, without series-specific synthesis or semantic routing.

Under Panel A's selected-coefficient protocol, Opus improves Average VUS-PR by $+0.0500$ over the Canonical Program, with a paired-bootstrap 95% CI of $[+0.0344, +0.0656]$. Haiku's paired interval lies below zero, Sonnet's narrowly crosses zero, and Opus's lies above zero.

The Exhaustive `shape_lib` control closely matches the Opus bank size (35.13 versus 34.90 columns per series) yet achieves 0.5016 Average VUS-PR. Thus, bank size and exhaustive coverage of the supplied primitives alone do not explain the Opus result.

Panel B applies the primary configuration's Tuning-48-selected $\alpha = 0.50$ to every detector bank. Under this shared final-fusion rule, Opus exceeds the Canonical Program by $+0.0452$ Average VUS-PR, with a paired-bootstrap 95% CI of $[+0.0321, +0.0589]$. Performance also increases from Haiku to Sonnet to Opus (0.4959, 0.5115, and 0.5498), showing that the compiler ordering is not an artifact of assigning different PostShare coefficients across banks.

4.4 Compiler-Stack Ablation

The detector-oriented scaffold, rather than prefix profiling alone, produces the dominant gain on Tuning-48. Profiling reduces mean input tokens to 11.3% of the raw-prefix condition with little change in VUS-PR, whereas adding the detector interface and program contract improves VUS-PR by $+0.1752$. Table 3 traces this compiler stack under identical downstream fusion.

4.5 What the Stronger Compiler Writes

Opus does not merely route among the supplied numerical primitives. Across three formal compilations on Tuning-48 and three on Eval-350, runtime metadata records 41,735 detector columns from 1,194 retained programs: 25,925 (62.1%) directly invoke a supplied `shape_lib` primitive, 1,451 (3.5%) fuse or summarize shape columns, and 14,359 (34.4%) are compiler-written detectors rather than direct library calls. Re-fused alone, neither provenance subset recovers the full bank: 0.4844 for the shape-only and 0.3628 for the compiler-only subset (Appendix A.2).

The compiler-written detectors span general-purpose point, change, residual, seasonal, and local-scale evidence, alongside recurring specialists tailored to distinctive prefix regimes: activity-rate

Table 4: Series 827 mechanism comparison. Occupancy matches two local-activity controls, while NMRL improves over unconditional dwell. A checkmark identifies standardized mechanisms inspired by recurring Opus-generated variants.

Signal	Score	LLMDP-U-inspired	VUS-PR
Persistence	Unconditional dwell	–	0.5389
	NMRL	✓	**0.8648**
Local activity	**Occupancy**	✓	**0.9962**
	Raw rolling mean	–	0.9977
	Activity intensity	–	0.9980

and persistence for background-dominated event processes, and value-support specialists (Appendix A.5). Together, they show how detector programming turns each reference-prefix profile into series-specific executable evidence that the fixed fusion rule scores without further LLM inference (profile-to-detector statistics in Appendix A.6).

Activity and persistence specialists. Series 827 (Exathlon 15) has a zero-dominated reference prefix (80.0%). Across three independent Opus compilations, generated programs repeatedly construct local activity-rate and nonbackground run-length variants. We standardize these mechanisms as Occupancy and nonmodal run-length (NMRL), respectively; Appendix A.4 gives the complete definitions. Table 4 compares them with matched controls.

Occupancy reaches 0.9962 VUS-PR, on par with raw rolling mean (0.9977) and activity intensity (0.9980): on Series 827, sustained local activity is the informative signal. NMRL isolates nonbackground-episode persistence and reaches 0.8648, versus 0.5389 for unconditional dwell. The persistence mechanism extends beyond Series 827: a prefix-only regime rule distilled a posteriori from the generated programs selects 17 series across seven datasets and four domains, where NMRL improves over unconditional dwell by $+0.1247$ mean VUS-PR. A recurring compiler-written mechanism thus survives standardization beyond the motivating case.

5 Discussion

Detector programming separates adaptation from execution. LLMDP-U performs profile-conditioned synthesis during compilation and stores its result as an executable detector program. Subsequent score computation is numerical, inspectable, and independently profileable. The retained programs carry over to later segments of the same series: executed on the prefix and only the late half of each series, Opus programs retain $+0.057$ over the Canonical Program, against $+0.061$ under full-series execution (Appendix A.3). The controls of Section 4.3 and Appendix A.2 locate the gain in the composition of library and compiler-written columns rather than in bank size, fusion coefficients, or primitive coverage, and performance increases from Haiku to Sonnet to Opus. This suggests a broader systems design principle for constrained time-series learning: foundation-model reasoning can be retained as an artifact rather than invoked at every scoring step.

Limitations and future work. The current instantiation assumes a reference prefix, uses offline transductive fusion, targets univariate series, and is evaluated on a single benchmark family. Compilation takes a median 113.0 s of LLM generation per series and each scoring pass 23.0 s of numerical execution (Appendix A.8). The modular interface, program contract, and validator provide explicit enforcement points for causal operators, runtime and bank-size budgets, and scenario-specific tests. Future work will develop low-latency, online, and multivariate variants and reusable programs across related series, domains, and sensors.

6 Conclusion

LLMDP-U recasts an LLM from an anomaly judge into a detector compiler. On Eval-350, the resulting programs outperform a matched no-LLM Canonical Program, compete with leading pretrained methods, and retain a gain that a matched-size exhaustive primitive bank does not reproduce. Provenance analysis further reveals recurring profile-conditioned mechanisms beyond direct library calls. The result is a persistent detector artifact that preserves adaptive reasoning while removing the LLM from numerical execution.

References

[1] Sarah Alnegheimish, Linh Nguyen, Laure Berti-Equille, and Kalyan Veeramachaneni. Large language models can be zero-shot anomaly detectors for time series? *arXiv preprint arXiv:2405.14755*, 2024.

[2] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the Language of Time Series. *Transactions on Machine Learning Research*, 2024. URL `https://arxiv.org/abs/2403.07815`.

[3] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. USAD: UnSupervised Anomaly Detection on Multivariate Time Series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 3395–3404, 2020. doi: 10.1145/3394486.3403392.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] Manqing Dong, Hao Huang, and Longbing Cao. Can LLMs serve as time series anomaly detectors? *arXiv preprint arXiv:2408.03475*, 2024.

[6] Utsav Dutta, Gerardo Pastrana, Sina Khoshfetrat Pakazad, and Henrik Ohlsson. Giving Sensors a Voice: Multimodal JEPA for Semantic Time-Series Embeddings. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306, 2026. URL `https://arxiv.org/abs/2605.31580`.

[7] Vijay Ekambaram, Subodh Kumar, Arindam Jati, Sumanta Mukherjee, Tomoya Sakai, Pankaj Dayama, Wesley Gifford, and Jayant Kalagnanam. TSPulse: Tiny Pre-Trained Models with Disentangled Representations for Rapid Time-Series Analysis. In *International Conference on Learning Representations*, 2026. URL `https://arxiv.org/abs/2505.13033`.

[8] 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.

[9] Zelin He, Sarah Alnegheimish, and Matthew Reimherr. Harnessing Vision-Language Models for Time Series Anomaly Detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(26):21690–21698, 2026. doi: 10.1609/aaai.v40i26.39319. URL `https://arxiv.org/abs/2506.06836`.

[10] Hyeongwon Kang, Jeongseob Kim, Jinwoo Park, and Pilsung Kang. Detecting Time Series Anomalies Like an Expert: A Multi-Agent LLM Framework with Specialized Analyzers. *arXiv preprint arXiv:2605.05725*, 2026.

[11] Uzair Khan, Luigi Capogrosso, Francesco Biondani, Michele Magno, Franco Fummi, Francesco Setti, and Marco Cristani. ChronosAD: Leveraging Time Series Foundation Models for Accurate Anomaly Detection. In *2026 IEEE 24th International Conference on Industrial Informatics (INDIN)*, 2026. URL `https://arxiv.org/abs/2606.01300`.

[12] 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. *arXiv preprint arXiv:2509.21190*, 2025.

[13] 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*, pages 4623–4634, 2025. doi: 10.1145/3711896.3737239. URL `https://arxiv.org/abs/2405.15370`.

[14] 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 37, Datasets and Benchmarks Track*, pages 108231–108261, 2024. doi: 10.52202/079017-3437.

[15] Qinghua Liu, Sam Heshmati, Zheda Mai, Zubin Abraham, John Paparrizos, and Liu Ren. MLLM4TS: Leveraging Vision and Multimodal Language Models for General Time-Series Analysis. *arXiv preprint arXiv:2510.07513*, 2025.

[16] John Paparrizos and Luis Gravano. k-Shape: Efficient and Accurate Clustering of Time Series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1855–1870, 2015. doi: 10.1145/2723372.2737793.

[17] Jun Woo 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 38*, pages 114115–114162, 2025. doi: 10.52202/085713-3444.

[18] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 427–438, 2000.

[19] Shebuti Rayana and Leman Akoglu. Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In *Proceedings of the 2015 SIAM International conference on data mining*, pages 622–630. SIAM, 2015.

[20] Peter J Rousseeuw and Christophe Croux. Alternatives to the median absolute deviation. *Journal of the American Statistical association*, 88(424):1273–1283, 1993.

[21] 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*, pages 2828–2837, 2019.

[22] 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.

[23] TheDatumOrg. TSB-AD Univariate Leaderboard Result Files. GitHub repository, commit e0975a5f7d3e65ab77e9fab24d1b5b51acda8f48, 2026. URL `https://github.com/TheDatumOrg/TSB-AD/tree/e0975a5f7d3e65ab77e9fab24d1b5b51acda8f48/benchmark_exp/leaderboard_results`. Snapshot retrieved September 3, 2026.

[24] 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. URL `https://arxiv.org/abs/2110.02642`.

[25] Xiongxiao Xu, Haoran Wang, Yueqing Liang, Philip S Yu, Yue Zhao, and Kai Shu. Can Multimodal LLMs Perform Time Series Anomaly Detection? In *Proceedings of the ACM Web Conference 2026*, pages 5392–5403, 2026. doi: 10.1145/3774904.3792376. URL `https://arxiv.org/abs/2502.17812`.

[26] John Yang, Carlos 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 37*, pages 50528–50652, 2024. doi: 10.52202/079017-1601. URL `https://arxiv.org/abs/2405.15793`.

[27] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1317–1322, 2016. doi: 10.1109/ICDM.2016.0179.

[28] Haoting Zhang and Shekhar Jain. LLM-Assisted Logic Rule Learning: Scaling Human Expertise for Time Series Anomaly Detection. *arXiv preprint arXiv:2601.19255*, 2026.

[29] Junru Zhang, Lang Feng, Haoran Shi, Xu Guo, Han Yu, Yabo Dong, and Duanqing Xu. AnomSeer: Reinforcing Multimodal LLMs to Reason for Time-Series Anomaly Detection. *arXiv preprint arXiv:2602.08868*, 2026.

[30] Yue Zhao, Zain Nasrullah, Maciej K Hryniewicki, and Zheng Li. LSCP: Locally selective combination in parallel outlier ensembles. In *Proceedings of the 2019 SIAM international conference on data mining*, pages 585–593. SIAM, 2019.

[31] Xiaona Zhou, Muntasir Wahed, Tianjiao Yu, Constantin Brif, and Ismini Lourentzou. Tiny but Trusted: Efficient Vision-Language Reasoning for Time-Series Anomaly Detection. *arXiv preprint arXiv:2605.30344*, 2026.

[32] Zihao Zhou and Rose Yu. Can LLMs Understand Time Series Anomalies? In *International Conference on Learning Representations*, 2025. URL `https://arxiv.org/abs/2410.05440`.

A Additional Experimental Details

A.1 Compilation and Reproducibility Details

Each series is compiled from its benchmark-provided reference prefix without raw samples or anomaly labels. The prompt contains a prefix-profile JSON object, numerical detector interface, and program contract, and requests a `CONFIG` dictionary and

```
detect_anomalies(series, config) -> (sanity_flag, detector_bank)
```

where `detector_bank` contains separate continuous detector columns and `sanity_flag` is used only during validation.

Profile schema. The prefix profile records marginal and robust distribution statistics, robust-STL trend and seasonal descriptors, FFT and autocorrelation periods, change-point segments, and a recurrence-derived candidate-window grid. Numeric prefix length and candidate scales are included in the prompt; the complete series is accessed only after a program has been retained. The prompt carries no dataset or series identifiers, anomaly labels, or raw samples; the profile is the compiler's only series-specific input.

Generation and validation. The primary compiler is Claude Opus 5 with adaptive reasoning at high effort. Retained programs execute in Python 3.12 under the documented scientific Python allowlist. Generated code may not perform file access, subprocess execution, dynamic code execution, ML training, internal voting, or detector thresholding. Validation checks syntax, function presence, imports, forbidden operations, execution, output length, and numerical detector-bank validity. It also runs the auxiliary sanity flag on a synthetic spike series and a constant series. Invalid generations receive the code and validation error as feedback, for at most ten total generation attempts. Validation enforces a 60-second budget on the sanity checks and a 600-second budget on full-series execution; no bank-size cap is imposed.

Numerical interface and controls. The fixed `shape_lib` interface provides matrix-profile discord, dictionary k-nearest-neighbor distance, template normalized cross-correlation, subsequence PCA reconstruction, cycle residual, and shape changepoint. The Canonical Program replaces compilation with a fixed heterogeneous bank of robust point, residual, seasonal, and subsequence scores. The Exhaustive `shape_lib` control instead instantiates all six supplied primitives at every eligible profiler-derived scale, without semantic routing. Raw Prefix and Profile Only (Table 3) prompt the same compiler to write a detector program from the serialized reference prefix and from its numeric profile, respectively, without the `shape_lib` interface or program contract.

A.2 Column-Provenance Subsets of the Opus Banks

Each persisted Opus bank on Eval-350 is split by the runtime provenance of its columns (Section 4.5): the *shape-only* subset retains the direct `shape_lib` calls and their summaries, and the *compiler-only* subset retains the compiler-written columns. Each subset is re-fused with the frozen rule at $\alpha = 0.50$ and compared with the full bank and the Canonical Program under the same coefficient (Table A.1). A series whose subset is empty receives its Canonical Program score, so that all arms are compared on the same 350 series. Neither subset recovers the full-bank result.

Table A.1: Column-provenance subsets of the Opus banks on Eval-350 under shared $\alpha = 0.50$ fusion. Δ and 95% CIs are paired against the Canonical Program; Opus values are means $\pm$ standard deviation across three runs.

Experiment	α	Bank size	VUS-PR ↑	Δ	95% CI
Canonical Program	0.50	21.52	0.5046	–	–
LLMDP-U, shape-only	0.50	22.9	0.4844 ± 0.0034	-0.0202	$[-0.0330, -0.0071]$
LLMDP-U, compiler-only	0.50	12.0	0.3628 ± 0.0017	-0.1418	$[-0.1810, -0.1021]$
LLMDP-U, full bank	0.50	34.90	0.5498 ± 0.0023	$+0.0452$	$[+0.0321, +0.0589]$

A.3 Reuse on a Later Segment without Intervening Data

To test whether a retained program can be reused on data that arrives after compilation, without access to the intervening history, we split the post-prefix region of each Eval-350 series at its midpoint into an early half and a late half of length L, and re-execute the retained program on the reference prefix followed only by the late half; the frozen fusion rule ($\alpha = 0.50$) is applied to the resulting bank, so neither execution nor fusion observes the omitted early half. The Canonical Program is executed on the same segment from its prefix-derived profile. Concatenating the prefix and the late half creates an artificial discontinuity at the join: windows that straddle it mix prefix values with late-half values, and window-based detectors respond for up to one window length afterwards. We therefore discard the first B points after the join before scoring, with $B = \min(w_{\max}, \lfloor L/4 \rfloor)$, where $w_{\max}$ is the largest candidate window length in the prefix profile; the cap keeps short late halves evaluable. The reference arm reuses the detector bank saved from the main full-series run, fused as in the main experiment, and is scored on the same late-half positions. VUS-PR is undefined when the evaluated region contains no anomaly, leaving 258 series on which all arms are defined. Table A.2 reports the results with and without the seam buffer.

Table A.2: Reuse on a later segment: late-half Average VUS-PR on the 258 comparable Eval-350 series. Panel A: Opus programs executed on the full series; Panel B: Opus programs re-executed on the prefix and late half only; Panel C: as Panel B, scored without the seam buffer. Δ and 95% CIs are paired against the Canonical Program within each panel; Opus values are means $\pm$ standard deviation across three runs.

Experiment	α	VUS-PR ↑	Δ	95% CI
Panel A: Full-series execution				
Canonical Program	0.50	0.5480	–	–
LLMDP-U	0.50	0.6087 ± 0.0010	$+0.0607$	$[+0.0433, +0.0787]$
Panel B: Prefix + late half only				
Canonical Program	0.50	0.5442	–	–
LLMDP-U	0.50	0.6011 ± 0.0012	$+0.0569$	$[+0.0393, +0.0752]$
Panel C: Prefix + late half only, no seam buffer				
Canonical Program	0.50	0.5262	–	–
LLMDP-U	0.50	0.5829 ± 0.0039	$+0.0567$	$[+0.0396, +0.0744]$

A.4 Activity and Persistence Mechanisms

Series 827 spends 80.0% of its reference prefix at zero. Let $P = \{1, \ldots, t_0\}$ denote the reference prefix and $\mathbf{x}_P = (x_t)_{t \in P}$ its observed sequence. We treat zero as background and define

$$u_t = \mathbf{1}[x_t \neq 0].$$

Here, u_t indicates activity. For any time-indexed sequence v, let W_t be the boundary-truncated centered window of width w, and define its local mean as

$$m_w(v)_t = \frac{1}{|W_t|} \sum_{i \in W_t} v_i.$$

For statistic q_t and prefix reference $\mathbf{q}_P$, define the robust MAD calibration [20]

$$\mathcal{Z}(q_t \mid \mathbf{q}_P) = \frac{q_t - \mathrm{med}(\mathbf{q}_P)}{1.4826\ \mathrm{MAD}(\mathbf{q}_P) + \epsilon}.$$

For rolling statistics, $m_w^P(v)$ denotes the local means recomputed after truncating v at the prefix boundary. Absolute values represent two-sided deviations, while $[\cdot]_+$ retains unusually large values. Define $a_t = |\mathcal{Z}(x_t \mid \mathbf{x}_P)|$.

Let L_t be the length of the maximal constant run of u containing t. Multiplication by u_t restricts NMRL to active, nonbackground runs. If ℓ_r is the length of run r, define

$$\mathbf{r}_P = \big(\log(1+\ell_r)\big)_{r:\, r \text{ is an active run completed within } P},$$
$$\mathbf{d}_P = \big(\log(1+\ell_r)\big)_{r:\, r \text{ is any run completed within } P}.$$

When $|\mathbf{r}_P| < 4$, NMRL is set to zero. Informally, NMRL measures whether the current active episode lasts unusually long relative to active episodes in the prefix, and Occupancy whether the local fraction of nonbackground samples is unusually high relative to the prefix. Table A.3 defines the compared scores, and Figure A.1 shows their responses on Series 827.

Table A.3: Activity and persistence comparison on Series 827. Unconditional dwell is the matched persistence control; raw rolling mean and activity intensity are local-activity controls. All scores use prefix-only robust calibration, and rolling scores use $w = 161$.

Score	Definition	VUS-PR
Unconditional dwell	$\lvert\mathcal{Z}(\log(1+L_t) \mid \mathbf{d}_P)\rvert$	0.5389
NMRL	$u_t\,[\mathcal{Z}(\log(1+L_t) \mid \mathbf{r}_P)]_+$	**0.8648**
Occupancy	$\lvert\mathcal{Z}(m_w(u)_t \mid m_w^P(u))\rvert$	**0.9962**
Raw rolling mean	$\lvert\mathcal{Z}(m_w(x)_t \mid m_w^P(x))\rvert$	0.9977
Activity intensity	$\big[\mathcal{Z}(m_w(a)_t \mid m_w^P(a))\big]_+$	0.9980

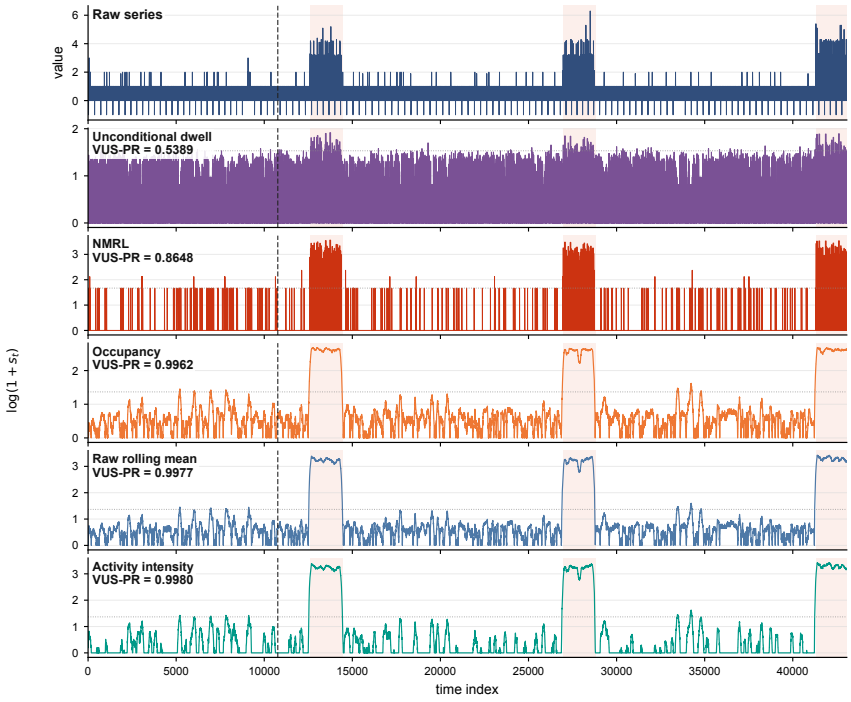


Figure A.1: Series 827 activity and persistence comparison, including unconditional dwell as the matched persistence control, ordered by VUS-PR. Detector panels show $\log(1+s_t)$. Occupancy and NMRL standardize recurring Opus-generated mechanisms; raw rolling mean and activity intensity are local-activity controls. Red shading marks anomalies; the dashed line marks the prefix boundary.

A series enters the exact-modal regime when its reference prefix, rounded to eight decimal places, contains at most 128 distinct values with modal mass of at least 65%; the modal value is background and all others are activity. These definitions were standardized post hoc from the generated programs after the main results were obtained: the run-count floor follows the minimum-run fallback used in the programs themselves, and the cardinality and modal-mass gates were set once to favor precision over recall.

The exact-modal regime contains 17 distinct series across seven datasets and four domains. Under the official TSB-AD evaluation, NMRL reaches 0.2882 mean VUS-PR, compared with 0.1635 for unconditional dwell, for a mean paired gain of $+0.1247$. A 10,000-replicate paired nonparametric bootstrap over the 17 series gives a 95% interval of $[+0.0387, +0.2235]$. Table A.4 summarizes the aggregate comparison, and Table A.5 reports the per-series results.

Table A.4: Scoped NMRL comparison in the exact-modal regime using official TSB-AD VUS-PR.

Metric	VUS-PR	Δ	95% CI
Dwell	0.1635	–	–
NMRL	**0.2882**	**+0.1247**	**[+0.0387, +0.2235]**

Table A.5: Per-series official VUS-PR for the exact-modal NMRL comparison.

Dataset	Series ID	NMRL	Dwell	Δ
Exathlon	3	0.6137	0.1055	+0.5082
Exathlon	8	0.9725	0.9713	+0.0012
Exathlon	15	0.8648	0.5389	+0.3259
IOPS	6	0.1099	0.0006	+0.1093
MSL	1	0.0717	0.0717	+0.0000
MSL	5	0.0383	0.0906	-0.0523
NAB	3	0.1341	0.1132	+0.0209
NAB	8	0.3305	0.0967	+0.2338
NAB	17	0.1147	0.0926	+0.0221
NEK	1	0.4915	0.5536	-0.0621
SMAP	1	0.0185	0.0185	+0.0000
SMAP	4	0.0132	0.0132	+0.0000
SMAP	8	0.3883	0.0213	+0.3670
SMAP	12	0.0149	0.0261	-0.0112
SMAP	15	0.0373	0.0375	-0.0002
SMD	3	0.5939	0.0050	+0.5889
SMD	11	0.0917	0.0228	+0.0689

A.5 Illustrative Value-Support Cases

Opus also constructs value-support specialists. Instead of measuring deviation from a global center, these detectors score how far an observation falls from values supported by the reference prefix. We express this recurring idea as Exact Support. Let

$$z_t = \frac{x_t - \mathrm{med}(x_{1:t_0})}{1.4826\ \mathrm{MAD}(x_{1:t_0}) + \epsilon}, \qquad s_t^{\mathrm{support}} = \min_{i \le t_0} |z_t - z_i|.$$

Exact Support is a prefix-conditioned nearest-neighbor novelty score [18]. For comparison, let μ_1, μ_2 be deterministic one-dimensional k-means centers fitted on the normalized prefix and define

$$s_t^{\mathrm{2m}} = \min_{k \in \{1,2\}} |z_t - \mu_k|.$$

UCR-392 and UCR-351 illustrate this mechanism: their anomalous segments introduce values weakly supported by the reference prefix and are detected by Exact Support but missed by nearest two-means. On UCR-392, Exact Support reaches 0.7574 VUS-PR, versus 0.0093 for nearest two-means. On UCR-351, it reaches 0.4534, versus 0.0114. Figure A.2 visualizes the anomaly regions and detector ranks for both cases.

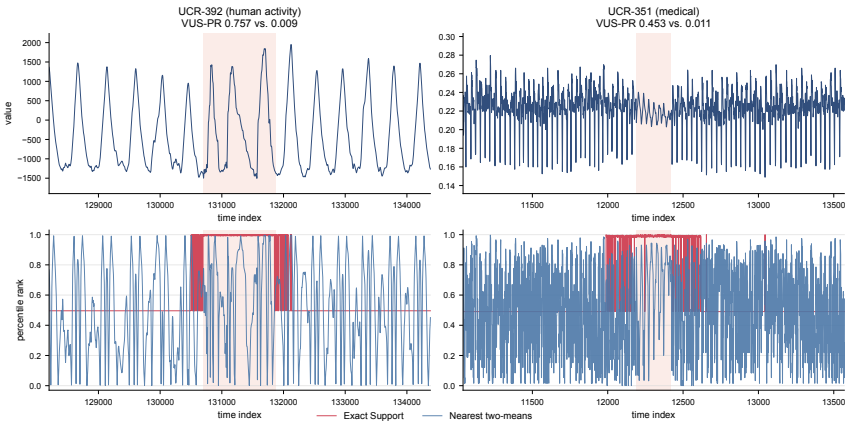


Figure A.2: Two illustrative value-support cases. Top: anomaly-region zooms for UCR-392 and UCR-351. Bottom: tie-aware percentile ranks for Exact Support and nearest two-means. Red shading marks anomalies.

A.6 Profile-Conditioned Construction in the Audited Programs

The prefix profile is the compiler's only series-specific input. To characterize how its fields shape the synthesized detectors, we group the 12,624 compiler-written numerical columns of the 1,050 retained Eval-350 programs (three Opus runs; Section 4.5) into three families by runtime column name—seasonal or cycle residual, level or change-point, and activity-rate or run-length—and report, for each profile condition, the share of programs containing at least one column of the corresponding family (Table A.6). The analysis is descriptive and was conducted after the main results.

Two profile fields are associated with which detectors are written. Level and change-point detectors appear in 68% of programs whose prefix profile reports no change points and in 95% of those reporting one or more. Activity-rate and run-length detectors appear in 84% of programs whose prefix quantiles collapse ($p_{25} = p_{75}$; 51 programs, the discrete background-dominated regime of Appendix A.4) and in 13% otherwise. Seasonal residual detectors, by contrast, are nearly universal across periodicity-confidence levels: the prompt's worked example is itself a seasonal series and its guidelines suggest seasonal decomposition, so their presence reflects the scaffold rather than the profile. Their scale, however, tracks the profile: among the 32.5% of programs whose seasonal columns carry a numeric period tag, the tag coincides with a period reported in the profile's decomposition, FFT, or autocorrelation fields in 99.4% of cases. Series 551 (YAHOO) illustrates the pattern. The program header restates the profile (seasonal strength 0.997, dominant period ≈ 100–104, negative kurtosis indicating bimodal plateaus), and the program emits `phase_resid_z_p100`, `phase_resid_z_p104`, and `stl_resid_z_p104` alongside local contextual residuals and the supplied shape primitives.

Table A.6: Share of retained Eval-350 Opus programs (three runs, $n = 1{,}050$) containing at least one compiler-written column of the indicated family, by prefix-profile condition.

Profile field	Condition	Compiler-written family	Programs
Periodicity confidence	low	seasonal / cycle residual	88.9%
	medium	seasonal / cycle residual	99.6%
	high	seasonal / cycle residual	99.6%
Change points in prefix	0	level / change-point	67.6%
	1–4	level / change-point	95.7%
	≥ 5	level / change-point	94.1%
Prefix quantiles	$p_{25} = p_{75}$	activity-rate / run-length	84.3%
	$p_{25} \neq p_{75}$	activity-rate / run-length	12.6%

A.7 Per-Dataset Comparison with the Canonical Program

Table A.7 reports Eval-350 Average VUS-PR by source dataset for Opus-compiled programs (three-run mean) and the Canonical Program under the shared $\alpha = 0.50$ fusion rule. The compiled advantage is broad but not uniform: it is largest on finance, synthetic, and server-monitoring datasets (Stock, MGAB, Exathlon, SMD) and most negative on three physiological-signal datasets (OPPORTUNITY, MITDB, LTDB), where the Canonical Program is already strong.

Table A.7: Per-dataset Average VUS-PR on Eval-350 under shared $\alpha = 0.50$ fusion. Opus is the three-run mean; Δ is Opus minus Canonical.

Dataset	n	Opus	Canon.	Δ	Dataset	n	Opus	Canon.	Δ
Stock	8	0.770	0.626	$+0.144$	Daphnet	1	0.460	0.426	$+0.034$
MGAB	8	0.272	0.146	$+0.127$	CATSv2	1	0.364	0.332	$+0.032$
Exathlon	30	0.773	0.655	$+0.118$	MSL	7	0.465	0.437	$+0.029$
SMD	33	0.623	0.512	$+0.111$	NAB	23	0.381	0.372	$+0.010$
WSD	20	0.323	0.242	$+0.081$	SMAP	17	0.700	0.690	$+0.009$
NEK	8	0.492	0.428	$+0.065$	Power	1	0.432	0.425	$+0.007$
IOPS	15	0.351	0.290	$+0.061$	SWaT	1	0.626	0.634	-0.008
YAHOO	30	0.763	0.711	$+0.053$	SED	2	0.844	0.856	-0.012
TODS	13	0.886	0.839	$+0.046$	TAO	2	0.939	0.951	-0.013
SVDB	18	0.634	0.591	$+0.043$	OPPORTUNITY	27	0.307	0.369	-0.062
UCR	70	0.484	0.447	$+0.037$	MITDB	7	0.452	0.520	-0.068
					LTDB	8	0.524	0.595	-0.071

A.8 Compilation Reliability and Runtime

Across three Eval-350 compilations, LLMDP-U retained all $1{,}050$ requested detector programs within ten generation attempts. The median number of attempts was 2 (IQR 1–4), and 392 programs (37.3%) compiled on the first attempt.

Cumulative LLM generation sums model-call latency across outer attempts and excludes profiling, validation, and detector execution. Table A.8 summarizes compilation reliability and runtime, and Figure A.3 shows the corresponding latency distributions.

Table A.8: Compilation reliability and runtime summary.

Measurement	Scope	Median (IQR)	95th percentile
Generation attempts	$3\times$ Eval-350	2 (1–4)	9
Cumulative LLM generation	$3\times$ Eval-350	113.0 (58.4–213.6) s	437.4 s
Retry-normalized LLM call	$3\times$ Eval-350	52.3 (48.0–57.0) s	66.8 s
Numerical execution	Eval-350 isolated replay ($n = 350$)	23.0 (18.1–31.5) s	81.6 s

To isolate per-program latency from cross-program contention, we replayed all 350 retained programs sequentially on an AWS m7i.24xlarge instance with 96 vCPUs and 384 GiB of memory, using one outer worker and one active program at a time. The execution environment exposed 96 Numba threads to each program, while BLAS thread pools were restricted to one thread. Timings include input serialization, process startup, detector computation, and output parsing, but exclude compilation and validation. All programs completed. Median latency was 23.0 seconds (IQR 18.1–31.5), the 95th percentile was 81.6 seconds, and the maximum was 239.2 seconds. The complete replay required 2.84 hours of wall time.

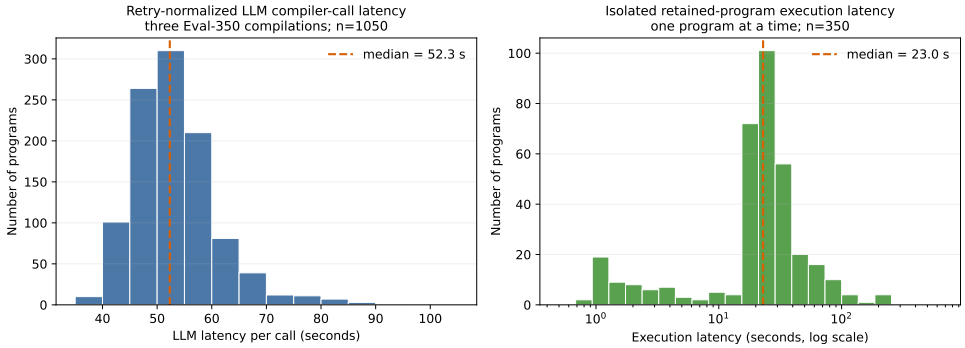


Figure A.3: LLM generation and numerical execution latency. Left: retry-normalized Claude Opus 5 generation latency across $1{,}050$ programs from three Eval-350 compilations. Right: isolated numerical execution latency for all 350 retained programs, replayed sequentially with one outer worker. The execution axis is logarithmic.

A.9 Fusion Selection on Tuning-48

LLMDP-U selects both fusion components exclusively on Tuning-48 and holds them fixed for Eval-350. Table A.9 compares aggregation rules with PostShare disabled: global top-$\lceil\sqrt{M}\rceil$ mean achieves 0.5428 Average VUS-PR, outperforming the strongest fixed-k alternative at 0.5082. Table A.10 then fixes this aggregation rule and varies the PostShare coefficient; $\alpha = 0.50$ gives the highest three-run mean of 0.5646 ± 0.0144.

Table A.9: Aggregation selection on Tuning-48.

Aggregation rule	Average VUS-PR
Rank maximum	0.3095
Rank mean	0.3859
Top-2 mean	0.4322
Top-3 mean	0.4816
Top-4 mean	0.5082
Global top-$\lceil\sqrt{M}\rceil$ mean	**0.5428**

Table A.10: PostShare selection on Tuning-48.

α	Average VUS-PR
0.00	0.5428 ± 0.0058
0.25	0.5645 ± 0.0115
0.50	$\mathbf{0.5646 \pm 0.0144}$
0.75	0.5634 ± 0.0141
1.00	0.5568 ± 0.0241

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Sections 1 and 4 state the contribution, evaluation setting, controlled comparisons, and main results. Section 5 bounds the scope of these claims.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 discusses the reference-prefix assumption, offline transductive fusion, univariate scope, compilation cost, execution latency, and future online and multivariate extensions.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper is empirical and makes no theoretical, theorem, or formal proof claims.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Sections 3–4 and Appendix A specify the public benchmark and evaluator, compiler inputs and contract, validation, fusion, controls, and repeated-run protocol. Because the hosted LLM is stochastic, the reproducibility target is protocol-level aggregate performance rather than byte-identical generated programs.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: TSB-AD-U and its evaluator are public, but the LLMDP-U implementation and generated programs are not released with this submission.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Sections 3–4 and Appendix A specify the data splits, compiler and prompt inputs, numerical interface, validation budget, fusion, controls, evaluator, and repeated-run protocol. No time-series model training or optimizer is used.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports mean and sample SD over three compilations. Paired differences use 10,000 series-level bootstrap resamples after averaging repeated-compilation scores within each series.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix A reports hosted-LLM attempts and latency plus replay hardware, thread settings, latency distributions, and total wall time. No training is performed; provider-side hardware, energy, and FLOPs are not observable.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The study uses a public benchmark and involves no human-subject research, personal-data collection, or sensitive model release. It conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Retained programs may reduce recurring LLM inference and improve inspectability. False alarms or missed detections remain possible, so deployment requires domain validation, monitoring, and human oversight.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The work releases no high-risk generative model, scraped dataset, or other artifact requiring misuse safeguards.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: TSB-AD and its evaluator are cited from the official repository under Apache License 2.0, while constituent datasets retain their original terms. No third-party model weights are redistributed.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The submission releases no new dataset, pretrained model, or software artifact.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The study involves no crowdsourcing, participant recruitment, or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The study involves no human participants or new personal-data collection, so IRB review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The LLM is a core methodological component. Sections 3–4 and Appendix A describe its inputs, program contract, validation and retries, and removal from score execution after compilation.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.