ENTROPY-GUIDED ADAPTIVE DIAGNOSTICS: IMPROVING CODE REASONING VIA EFFICIENT GENERATIVE-TIME REFINEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) show great potential in code generation, yet achieving logical accuracy remains fundamentally challenging. Three difficulties compound this problem: (1) unlike mathematical reasoning, code correctness requires execution to verify, making test-based feedback expensive in both time and compute; (2) code has strong structural dependencies that must be respected during revision — a change to one block must preserve the logical invariants of surrounding blocks; and (3) while token-level entropy correlates with logical errors, this signal is unreliable in isolation: hard problems exhibit uniformly high entropy, long solutions exhibit low entropy in later blocks once the algorithmic strategy is fixed, and high entropy at a particular line may simply reflect a legitimate choice among equivalent algorithmic strategies rather than a logical flaw. To address these issues, we present Entropy-Guided Adaptive Diagnostics (EGAD). EGAD shifts the focus from external execution to endogenous generative-time monitoring. After code generation completes, EGAD computes token-level entropy aggregated to line-level and block-level means via AST decomposition, and identifies anomalous lines whose entropy exceeds τ times their enclosing block's mean. These entropy signals serve as *guides* rather than mandates: a Suggestion LLM analyzes each flagged line to determine whether it represents a genuine logical flaw before producing targeted revision suggestions. Our method requires no additional training, no parallel sampling, and integrates seamlessly into existing pipelines. Evaluated across HumanEval, LiveCodeBench, and MLE-bench on three model families, EGAD delivers consistent and substantial gains. On LiveCodeBench, EGAD lifts Qwen3-80B-A3B from 65.89% to **76.56%** and Qwen3-30B-A3B from 48.88% to **66.93%**. Under a matched inference budget, EGAD outperforms DeepThink (Fu et al., 2025) — which uses entropy-guided path selection — by up to 7.9pp on LiveCodeBench-lite, confirming that spatially-targeted line-level intervention is a more effective use of compute than path-level filtering alone.

1 INTRODUCTION

Code reasoning is a central capability of large language models, yet generating functionally correct programs remains fundamentally difficult due to the gap between syntactic plausibility and semantic correctness. The emergence of Chain-of-Thought (CoT) prompting (Wei et al., 2022) has substantially advanced multi-step reasoning, and Self-Consistency (SC) (Wang et al., 2022) further improved reliability by marginalizing over multiple sampled reasoning paths. Subsequent work has extended this paradigm through multi-agent debate (Du et al., 2023) and higher-order aggregation strategies (Ai et al., 2025). However, these advances were largely validated on mathematical and symbolic tasks, where answers are concise and aggregate-friendly; their transfer to code generation, where correctness depends on the coherent interplay of sequential logical operations, remains limited. Moreover, these approaches rely on massive parallel sampling, making inference-time accuracy gains computationally expensive.

Existing self-consistency methods for code generation suffer from a structural blind spot: they evaluate correctness only after generation is complete, leaving latent logical errors undetected until

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.

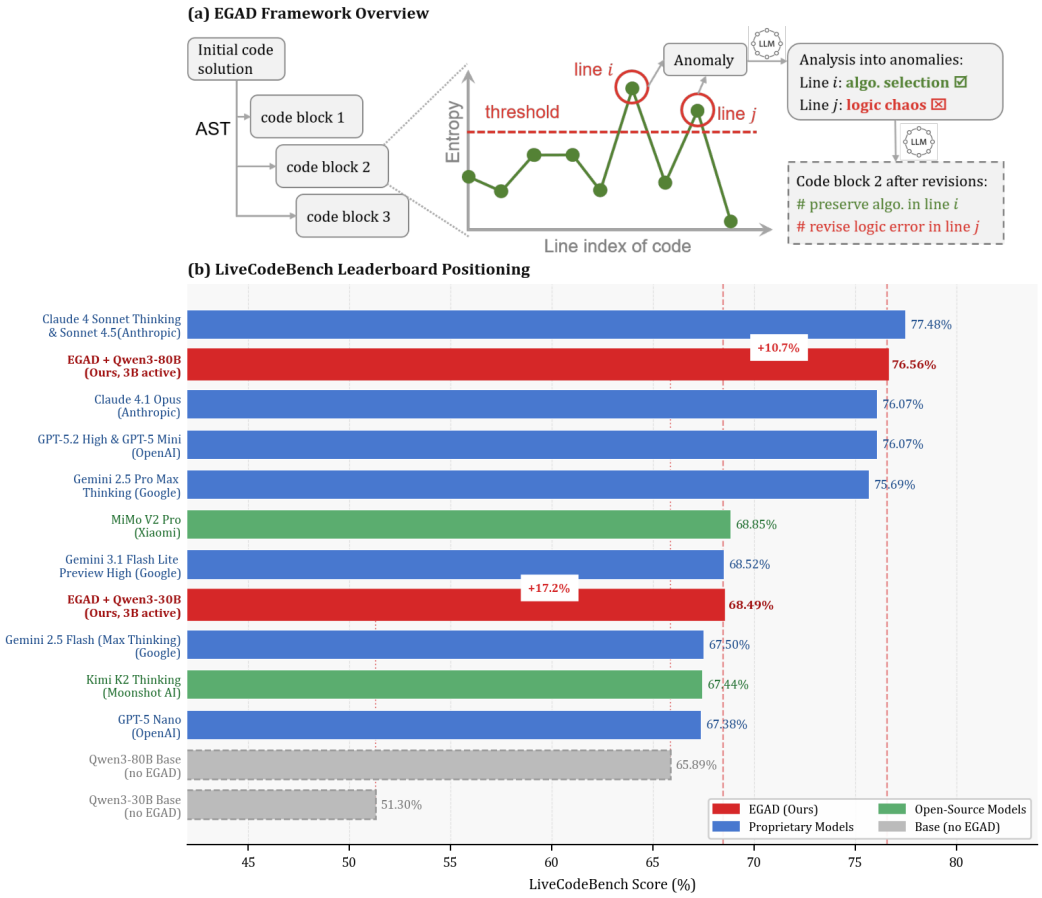


Figure 1: **(a)** EGAD framework. A Code LLM generates an initial solution; the Entropy Analysis module computes token-level entropy aggregated to line-level and block-level means via AST decomposition; the Suggestion LLM analyzes anomalous lines (high entropy: potential path collapse; low entropy with val↑/test↓: potential confident error) and produces targeted revision suggestions; the Code LLM, with entropy guidance, focuses on the corresponding blocks. **(b)** LiveCodeBench leaderboard positioning (source: livebench.ai, April 2026). EGAD lifts Qwen3-80B-A3B from 65.89% to **76.56%**, surpassing Claude 4.1 Opus (76.07%), GPT-5 Mini (76.07%), and Gemini 2.5 Pro Max Thinking (75.69%). EGAD lifts Qwen3-30B-A3B from 48.88% to **66.93%**, surpassing Devstral 2 (66.79%).

execution. Approaches that apply multi-perspective SC to code (Huang et al., 2024) or evaluate identity-based chains (Min et al., 2023) still rely on post-hoc aggregation of complete solutions. As demonstrated by recent studies on dynamic distributional alignment (Li et al., 2025) and reasoning-aware sampling (Wan et al., 2025), parallel sampling of full traces yields diminishing accuracy returns at prohibitive computational cost. The dominant "Generate-Test-Fix" paradigm compounds this problem: it requires external execution feedback that is expensive, unavailable in zero-shot settings, and vulnerable to the "Self-Consistent Hallucination" trap—where all sampled paths converge on the same logically flawed solution that achieves deceptively high validation scores through data leakage or hard-coded memorization.

The fundamental difficulty in applying uncertainty-based monitoring to code is twofold. First, code has strong structural dependencies: unlike natural language, a modification to one block must preserve the logical invariants established by surrounding blocks, and naive token-level interventions risk breaking syntactic or semantic coherence. Second, the entropy signal is inherently noisy in the code domain: problem difficulty causes uniformly elevated entropy across an entire solution; once an algorithmic strategy is established in early blocks, later blocks exhibit low entropy regardless

of correctness; and high entropy at a particular line may simply reflect a legitimate choice among equivalent algorithmic strategies rather than a logical flaw. These two properties mean that raw entropy cannot be used as a direct error indicator — it must be interpreted relative to the structural context of the code.

The most relevant prior work is **DeepThink** (Fu et al., 2025), which monitors model-internal confidence during generation to identify and filter low-quality reasoning traces; it serves as the primary baseline in our experiments. However, DeepThink requires up to 512 parallel samples to achieve high accuracy, making it computationally expensive. Subsequent works build on DeepThink's direction: EAGER (Scalena et al., 2025) branches at high-entropy tokens, EDIS (Zhu et al., 2026) tracks entropy trajectory dynamics, and CoRefine (Jin et al., 2026) uses a confidence trace to trigger self-refinement. However, all of these — including DeepThink — operate at the individual token level or over the full trace, with no awareness of code syntax or semantics. None can localize a specific code block as the source of a logical anomaly. The missing abstraction is the *functional code block* as the unit of entropy measurement — a granularity naturally aligned with Abstract Syntax Tree (AST) structure — which would enable unsupervised anomaly localization without labels, execution, or any external oracle.

We propose Entropy-Guided Adaptive Diagnostics (EGAD), a training-free framework that, after code generation completes, partitions the generated code into AST-aligned functional blocks and uses token-level entropy as a guide to detect and correct anomalous lines. Concretely, EGAD (1) partitions the generated code $\mathcal{C}$ into a sequence of functional blocks $\{B_1, \ldots, B_n\}$ via AST decomposition; (2) computes token-level entropy for each token, aggregated to line-level mean $H(l_{ij})$ for each line j in block B_i and block-level mean $\overline{H}(B_i)$; (3) flags line l_{ij} as a high-entropy anomaly when $H(l_{ij}) > \tau \cdot \overline{H}(B_i)$, where $\tau = 1.5$; and (4) initiates a spatially-targeted "Rethink" phase in which a Suggestion LLM analyzes each flagged line to determine whether it represents a genuine logical flaw — entropy serves as a guide rather than a mandate — and produces targeted revision suggestions for the corresponding block. Olausson et al. (2023) demonstrate that naive self-repair without targeted diagnostic signals frequently fails to improve correctness; EGAD addresses this by grounding the Rethink in precise entropy-localized evidence.

Our main contributions are threefold:

1. Entropy-Based Anomaly Detection: We propose a diagnostic paradigm that monitors token-level entropy aggregated to line-level and block-level means after code generation completes, requiring no parallel sampling, no execution feedback, and no additional training. By interpreting entropy relative to AST-aligned block structure, our method enables the unsupervised identification of latent logical inconsistencies — such as algorithmic logic deviations and subtle data leakage — that raw token-level entropy cannot reliably detect.
2. Entropy-Guided Targeted Intervention: We develop an entropy-guided "Rethink" mechanism triggered by line-level anomalies. Guided by entropy signals, EGAD provides the model with precise block-level diagnostic context, allowing the model to re-evaluate and rectify logic-critical components. Crucially, entropy serves as a guide rather than a mandate: the Suggestion LLM first analyzes whether a flagged line is a genuine error before producing revision suggestions.
3. Empirical Validation and Efficiency: We conduct extensive evaluations across general programming (HumanEval, LiveCodeBench) and complex data science (MLE-bench) benchmarks. Experimental results demonstrate that EGAD significantly enhances code generation accuracy and logical robustness. Notably, our framework is model-agnostic and requires no additional training, offering a resource-efficient path for robust code synthesis.

2 Related Work

Self-Consistency and Test-Time Scaling. Self-Consistency (Wang et al., 2022) established the paradigm of marginalizing over multiple sampled reasoning paths via majority voting. Subsequent work has extended this in several directions: multi-agent debate (Du et al., 2023) introduces adversarial cross-checking; higher-order aggregation (Ai et al., 2025) exploits inter-sample correlations beyond simple counts; and reasoning-aware sampling (Wan et al., 2025) prunes redundant paths to reduce inference cost. For code specifically, Huang et al. (2024) propose multi-perspective SC that

aggregates solutions from diverse prompting strategies, while Min et al. (2023) evaluate SC through identity-preserving chain transformations. More recent work has sought to improve the efficiency of parallel sampling: PETS (Liu et al., 2026) formulates trajectory allocation as a crowdsourcing optimization problem; SeerSC (Ji et al., 2025) uses a fast System-1 pass to pre-estimate answer entropy before committing to full System-2 sampling; and Xu et al. (2025) propose a semantic entropy-guided termination criterion for multi-round parallel reasoning. A common limitation of all these approaches is that they rely on the statistical aggregation of many parallel samples to achieve reasoning gains, making inference-time improvements computationally expensive. Furthermore, they operate on complete, post-hoc outputs and cannot localize errors within the generation process. Li et al. (2025) further show that naive parallel sampling suffers from distributional collapse, motivating intra-generation monitoring as pursued by EGAD.

Entropy and Confidence Signals for Generation Monitoring. The use of entropy as a reasoning signal has a growing body of support. Kuhn et al. (2023) introduce Semantic Uncertainty, which clusters semantically equivalent outputs before computing entropy, establishing that meaning-level rather than token-level entropy is a more reliable proxy for factual correctness — a direct conceptual predecessor of our block-level entropy aggregation approach. Kadavath et al. (2022) demonstrate that LLMs can self-assess their own uncertainty through sampling, providing theoretical grounding for using the model's own generative distribution as a diagnostic signal. Shorinwa et al. (2025) provide a comprehensive taxonomy of uncertainty quantification methods for LLMs. In the multi-step reasoning domain, Zhang et al. (2025) use entropy to guide exploration across reasoning branches, and Sharma & Chopra (2025) propose inverse-entropy voting to weight high-confidence paths.

Several recent works monitor uncertainty *during* generation rather than after. The pioneering work in this direction is **DeepThink** (Fu et al., 2025), which monitors model-internal confidence to identify and filter low-quality reasoning traces at test time; it serves as the primary baseline in our experiments. However, DeepThink requires up to 512 parallel samples to achieve high accuracy. Building on DeepThink, EAGER (Scalena et al., 2025) branches generation at high-entropy tokens, using token-level next-token entropy $H_t^{(K)}$ to trigger parallel path exploration. EDIS (Zhu et al., 2026) analyzes the temporal dynamics of the token-level entropy trajectory, identifying instability patterns (burst spikes and peak-valley rebounds) as predictors of incorrect reasoning. CoRefine (Jin et al., 2026) trains a lightweight Conv1D controller on the full-trace token confidence profile to decide whether to halt, rethink, or try an alternative approach. Vilas et al. (2025) extract latent-space trajectory signals from hidden-state representations to predict trace quality and enable early path selection. Despite their diversity, these methods share a critical structural gap: they all operate at the token level or over the full trace, with no awareness of code syntax or semantics. EAGER cannot distinguish uncertainty at a variable name from uncertainty at a data-loading call; EDIS's burst/rebound patterns are computed over dense token sequences with no notion of program structure; CoRefine's 16-bin temporal profile is semantics-free and triggers full-trace regeneration rather than spatially-targeted correction. EGAD addresses this gap by computing entropy at the *AST-aligned code-block* level — a granularity that is simultaneously coarser than individual tokens (enabling semantic grouping) and finer than the full trace (enabling precise anomaly localization), and that is uniquely suited to detecting code-specific failure modes such as data leakage and algorithmic paradigm drift.

Code Generation and Self-Repair. LLM-based code generation has advanced substantially through execution-guided feedback loops (Olausson et al., 2023), verbal reinforcement via Reflexion (Shinn et al., 2023), and modular self-revision through CodeChain (Le et al., 2023). Olausson et al. (2023) conduct a systematic study showing that naive self-repair without targeted diagnostic signals frequently fails to improve correctness, directly motivating EGAD's entropy-guided targeting. Reflexion (Shinn et al., 2023) accumulates verbal self-reflection across episodes but requires execution feedback and operates post-generation. CodeChain (Le et al., 2023) decomposes generation into sub-module chains with representative revision, which is structurally analogous to EGAD's block-level Rethink; however, CodeChain relies on explicit sub-module boundaries and post-hoc selection, whereas EGAD detects anomalous lines dynamically via token-level entropy without any predefined structure. Tree of Thoughts (Yao et al., 2023) enables deliberate search over reasoning paths but incurs exponential sampling cost; EGAD achieves targeted correction in linear time by using entropy spikes as precise intervention signals. Certaindex (Fu et al., 2024) monitors answer stabilization across reasoning steps to trigger early exit, but operates at the answer level rather than the code-block level and requires a verifier.

3 METHODOLOGY: ENTROPY-GUIDED ADAPTIVE DIAGNOSTICS

The core philosophy of **EGAD** is to transition from external execution-based verification to **endogenous generative-time monitoring**. We hypothesize that logical inconsistencies in code synthesis are preceded by measurable uncertainty in the model's internal state. By anchoring these uncertainty spikes, EGAD facilitates an entropy-guided intervention, as described in the following subsections. Figure 2 illustrates the complete pipeline on a concrete example.

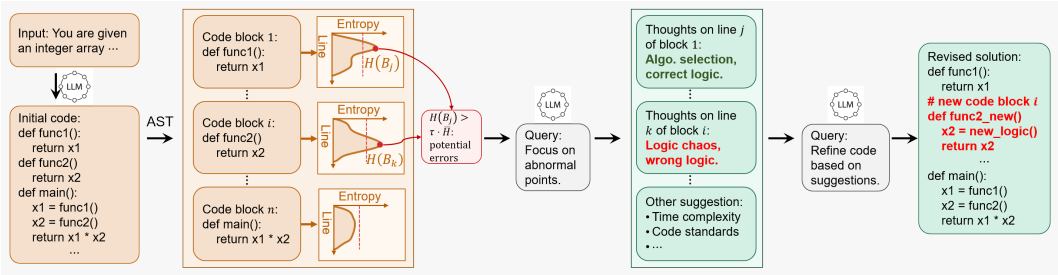


Figure 2: EGAD methodology pipeline. Given a problem description and an initial code solution, EGAD (1) decomposes the code into AST-aligned functional blocks (B_1, B_2, B_3); (2) computes token-level entropy aggregated to line-level and block-level means; (3) flags lines whose entropy exceeds $\tau \cdot \overline{H}(B_i)$ ($\tau = 1.5$) as high-entropy anomalies; (4) queries the Suggestion LLM with the original problem context, the full block decomposition, and the entropy signal to produce targeted diagnoses (e.g., "Logic chaos, wrong logic" for the corresponding block); and (5) queries the Code LLM to revise the corresponding block, guided by the entropy signal.

3.1 ENTROPY-BASED ANOMALY DETECTION

EGAD first partitions the generated code $\mathcal{C}$ into a sequence of functional blocks $\{B_1, B_2, \ldots, B_n\}$ using Python's `ast` module to extract top-level statement nodes. Each node, together with its nested sub-tree, constitutes one block B_i.

After generation completes, EGAD computes the token-level entropy $H(t)$ for each token t using the model's output probability distribution. These values are aggregated to obtain the **line-level mean entropy** $H(l_{ij})$ for each line j within block B_i, and the **block-level mean entropy**:

$$\overline{H}(B_i) = \frac{1}{|B_i|} \sum_j H(l_{ij}) \tag{1}$$

where $|B_i|$ denotes the number of lines in block B_i.

A line l_{ij} is flagged as a **high-entropy anomaly** when its entropy significantly exceeds the mean entropy of its enclosing block:

$$\text{Trigger}(l_{ij}) = \mathbb{I}\big(H(l_{ij}) > \tau \cdot \overline{H}(B_i)\big) \tag{2}$$

where $\tau = 1.5$ in all experiments. This block-relative threshold normalizes for two key confounders: (1) *problem difficulty* — hard problems exhibit uniformly elevated entropy across all blocks, so a global threshold would over-flag easy solutions and under-flag hard ones; (2) *code length* — once an algorithmic strategy is established in early blocks, later blocks exhibit low entropy regardless of correctness, so a global threshold would systematically miss errors in later blocks. By comparing each line against its own block's mean, EGAD identifies lines that are anomalous *relative to their local context*, which is a more reliable indicator of genuine logical uncertainty.

3.2 SPATIALLY-TARGETED "RETHINK" MECHANISM

Upon identifying high-entropy anomalies, EGAD initiates a targeted **Rethink** phase. Crucially, entropy serves as a *guide* rather than a mandate: the Suggestion LLM first analyzes each flagged line to determine whether it represents a genuine logical flaw, a legitimate algorithmic choice, or

an artifact of the code's structure. Only lines diagnosed as genuine errors receive targeted revision suggestions.

The Rethink prompt is constructed from three components: (1) the **original conversation context** $\mathcal{C}_{\text{orig}}$, which includes the problem description, constraints, and any prior dialogue; (2) the **full block decomposition** $\{B_1, \ldots, B_n\}$ of the generated solution, providing the model with a structured view of its own output; and (3) the **entropy signal** $\text{Signal}(H(l_{ij}))$, a natural-language description that identifies the corresponding flagged lines and their entropy values relative to the block mean.

The transformation from a flawed block B_i to a rectified block $\hat{B}_i$ is:

$$\hat{B}_i = \text{LLM}(\mathcal{C}_{\text{orig}} \oplus \{B_1, \ldots, B_n\} \oplus \text{Signal}(H(l_{ij}))) \tag{3}$$

where $\oplus$ denotes prompt concatenation. The entropy signal explicitly surfaces the localized uncertainty, breaking the model's **Mental Set** bias — the tendency to persist with a locally plausible but globally suboptimal strategy. As demonstrated in our experiments, this targeted intervention allows the model to shift from a suboptimal $O(N^3)$ heuristic to a mathematically optimal $O(1)$ implementation.

4 Experiment

4.1 Experimental Setup

Benchmarks. We evaluate on **HumanEval** (Chen et al., 2021), **LiveCodeBench (LCB)**, and **MLE-bench**. The first two are general code-generation benchmarks; MLE-bench evaluates autonomous data-science agents on Kaggle competitions and additionally exposes the reliability of their internal validation signals.

Baselines. We compare against four methods: (1) **Base**: direct greedy decoding with no test-time augmentation. (2) **DeepThink ($n = 3$)**: parallel sampling of three independent traces followed by majority voting, representative of standard test-time scaling (Fu et al., 2025). (3) **Reflect w.o. Entropy (Ablation)**: our full pipeline with the entropy-based anomaly detection replaced by unconditional self-reflection applied *globally after every block*, regardless of entropy level — this directly tests whether block-level entropy targeting is necessary, or whether indiscriminate global rethinking suffices, thereby isolating the contribution of the entropy signal from the Rethink mechanism itself. (4) **Ours (EGAD)**: the full system with entropy-guided anomaly detection and spatially-targeted Rethink.

Models. We evaluate across three model families of varying scale: Qwen3-Next-80B-A3B (80B MoE), Qwen3-30B-A3B (30B MoE), and Deepseek-Coder-16B (16B dense). The anomaly threshold is set to $\tau = 1.5$ uniformly across all models and benchmarks.

Metric. We report Pass@1 accuracy (%) for HumanEval and LiveCodeBench. For MLE-bench, we report Anomaly Ratio (frequency of cases where validation score exceeds test score), Score R^2 (correlation between internal validation and external test scores), and total Runtime (seconds).

We evaluate the performance of **EGAD** (Ours) against several baselines across three representative benchmarks. The primary results are summarized in Table 1.

4.2 Overall Performance

The experimental results demonstrate that **EGAD outperforms baselines in most settings** across various model scales and benchmarks.

Superiority over Base Models: Compared to vanilla inference (Base), EGAD achieves substantial accuracy gains. Notably, on the challenging LCB benchmark, EGAD improves the Pass@1 score of Deepseek-Coder-16B from 19.92% to 41.01%—a relative improvement of over 105%.

Advantage over Reflection-based Baselines: While standard Self-Reflection (Ablation) improves performance by re-evaluating the output, EGAD consistently provides an additional margin of 2.0% to 7.4% on LCB and LCB-lite. This confirms that entropy-guided diagnostic signals provide more **precise localization of logical flaws** than blind self-correction, which often suffers from stochastic trial-and-error.

Table 1: Main results on HumanEval (HE) and LiveCodeBench (LCB). We report the Pass@1 accuracy (%). For LCB and LCB-lite, we provide the breakdown across Easy (E), Medium (M), and Hard (H) challenges. $n = 3$ denotes parallel sampling for DeepThink. Best results are **bolded**.

Base LLM	Method	HE	LiveCodeBench (LCB)				LiveCodeBench-lite (V6)			
			E	M	H	Total	E	M	H	Total
Qwen3-Next -80B-A3B	Base	93.09	86.67	70.37	21.33	65.89	78.29	26.16	14.17	34.48
	DeepThink ($n = 3$)	94.51	86.67	73.02	24.00	67.68	82.17	33.97	16.67	37.90
	w.o. Entropy	96.34	90.00	79.90	36.00	74.48	90.70	**50.64**	22.08	47.43
	Ours	**96.75**	**90.00**	**81.45**	**42.67**	**76.56**	**93.80**	50.00	**24.58**	**49.14**
Qwen3-30B -A3B	Base	91.56	64.17	56.61	17.33	51.30	51.16	20.19	9.38	22.86
	DeepThink ($n = 3$)	**95.12**	77.50	62.96	24.00	59.90	75.97	26.28	10.42	31.24
	w.o. Entropy	**95.12**	79.17	66.14	**26.67**	62.76	84.88	31.73	11.25	35.43
	Ours	95.02	**85.83**	**70.37**	22.67	**66.93**	**90.70**	**36.54**	**13.13**	**39.14**
Deepseek-Coder-16B	Base	82.92	45.00	11.91	0.00	19.92	57.36	12.18	1.25	18.29
	DeepThink ($n = 3$)	85.56	52.50	12.70	0.00	22.66	61.14	13.46	1.25	19.62
	w.o. Entropy	94.51	68.75	23.81	2.00	33.60	73.64	27.56	7.92	29.90
	Ours	**95.12**	**75.00**	**33.34**	**6.00**	**41.02**	**79.07**	**31.41**	**8.75**	**32.76**

Positioning against the Official LiveCodeBench Leaderboard. To contextualize our results, we cross-reference with the official LiveCodeBench leaderboard (`https://livebench.ai`, accessed April 2026), which reports full-split LCB scores for a broad range of proprietary and open-source models. EGAD lifts Qwen3-Next-80B-A3B to **76.56%**, surpassing Claude 4.1 Opus (76.07%), GPT-5 Mini (76.07%), Gemini 2.5 Pro with Max Thinking (75.69%), and DeepSeek V3.2 (75.69%) on the same benchmark. For the smaller Qwen3-30B-A3B model (leaderboard base: 48.88%), EGAD achieves **66.93%**, surpassing Devstral 2 (66.79%), Grok 4.20 Beta (66.09%), and Gemini 2.5 Flash Lite with Max Thinking (66.41%), and approaching GPT-5 Nano (67.38%). We note that our evaluation split may differ slightly from the leaderboard's due to different problem collection windows; the comparisons are therefore directional rather than exact.

Robustness Across Model Scales. Our method exhibits strong scaling properties and robustness across different architecture sizes. **Large-scale Models (Qwen3-80B):** On the state-of-the-art 80B model, EGAD reaches a peak performance of 76.30% on LCB, surpassing DeepThink by 8.59%. This indicates that even highly capable models still possess "uncertainty blind spots" that EGAD can effectively identify and rectify through endogenous monitoring. **Mid-to-Small Models (Deepseek-16B):** The most significant gains are observed in smaller models. EGAD enables the 16B model to reach 95.12% on HumanEval, matching or even exceeding the performance of the much larger Qwen3-80B base model (93.09%). This suggests that EGAD can effectively **bridge the reasoning gap** for smaller parameter models through targeted refinement.

Efficiency vs. Performance (Comparison with DeepThink). A key highlight of our framework is its computational efficiency compared to standard test-time scaling methods. Crucially, the total number of LLM calls at inference time is **matched** between EGAD and DeepThink: DeepThink uses a warm-up phase of $n_{\text{warmup}} = 16$ samples to establish the entropy baseline, followed by 3 parallel traces; EGAD requires no warm-up phase and uses 3 sequential LLM calls (generate–diagnose–rethink). The performance difference therefore reflects the quality of the intervention strategy, not a difference in compute budget. **Performance Gap:** DeepThink relies on parallel sampling and majority voting, yet it underperforms EGAD in almost all LCB and LCB-lite scenarios. For instance, on Qwen3-30B (LCB-lite), EGAD (39.14%) outperforms DeepThink (31.24%) by a wide margin under the same inference budget. This demonstrates that **targeted entropy-guided intervention** is a more effective use of compute than undirected parallel sampling. **Observation on HumanEval:** On HumanEval, all methods show high performance, nearing saturation. While DeepThink slightly edges out other methods on the 30B model, EGAD maintains competitive results, proving its utility as a **resource-efficient reasoning enhancer** even on near-saturated benchmarks.

4.3 MLE-BENCH: AUTONOMOUS DATA SCIENCE AGENT EVALUATION

Beyond single-pass code generation, we evaluate EGAD within an autonomous data science pipeline using MLE-bench. A central challenge in automated machine learning is generalization collapse: agents often produce solutions that achieve high validation scores through over-fitting, but fail to

generalize to the test set. EGAD addresses this by monitoring structural entropy during the agent's initial drafting stage. Rather than relying on external execution metrics, EGAD treats localized entropy anomalies as indicators of potential reasoning divergence. With these signals, we force the Suggestion LLM to perform a rigorous logical audit of the generated pipeline—including data preprocessing and model partitioning, to ensure that the implementation is grounded in robust data science principles rather than stochastic pattern matching. Figure 3 illustrates this adapted diagnostic pipeline.

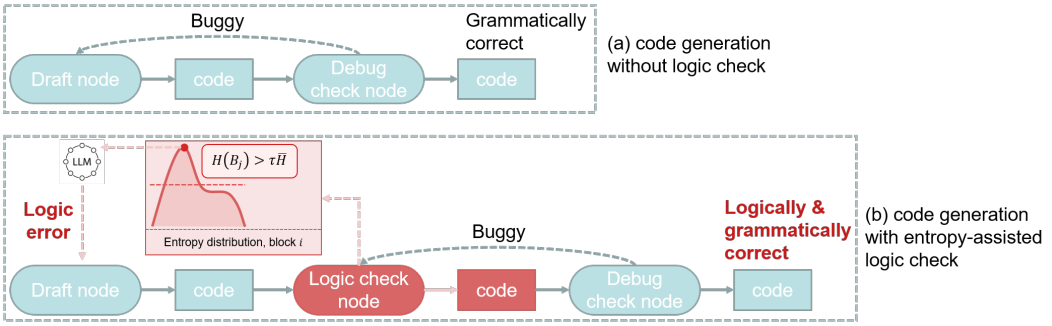


Figure 3: EGAD adapted for the autonomous data science agent setting. EGAD is applied at the agent's initial draft stage. Entropy anomaly of code lines triggers a Data Leakage Rethink. The Suggestion LLM receives both the entropy signal and the original context information, enabling it to distinguish genuine improvement from non-generalizable memorization.

We evaluate across four diverse Kaggle competitions: Twitter Sentiment Extraction, PetFinder Pawpularity, Whale Categorization, and LMSYS Chatbot Arena. We focus on three metrics: **Anomaly Ratio**, defined as the frequency of cases where validation score exceeds test score, **Score** R^2 measuring the coefficient of determination between validation and test set performance, and **Total Runtime**. The $\epsilon = 0.1$ threshold for identifying anomalies is a heuristic hyperparameter applied uniformly across all four tasks. To ensure a comprehensive and unbiased evaluation that is not solely dependent on this threshold, we also report the R^2 score as a complementary metric to characterize the global alignment between the agent's internal validation and its final test.

Table 2: Statistical performance across MLE-bench datasets. **Anomaly Ratio** and **Score** R^2 measure the reliability of validation scores. **Runtime** is reported in seconds.

Method	Twitter Sentiment			PetFinder Pawpularity			Whale Categorization			LMSYS Chatbot Arena		
	Anom. ↓	R^2 ↑	Time ↓	Anom. ↓	R^2 ↑	Time ↓	Anom. ↓	R^2 ↑	Time ↓	Anom. ↓	R^2 ↑	Time ↓
Base	75.7%	-5.02	77460	18.8%	-44.87	18380	55.0%	-27.02	31908	**11.1%**	-9.05	**34297**
Ablation	61.9%	-3.91	77083	18.8%	**0.95**	25623	31.6%	-22.74	83252	12.5%	-0.16	59243
Ours	**50.0%**	**0.75**	**36010**	**0.0%**	-2.06	**10660**	**26.3%**	**-10.5**	**28525**	12.5%	**0.99**	48249

4.3.1 Identification of Validation-Test Discrepancies

As shown in Table 2, EGAD significantly reduces the anomaly ratio. In the *Twitter Sentiment* competition, EGAD lowers the anomaly rate from 75.7% to 50.0% by identifying leakage-prone operations, such as "memory-based" lookup logic, and replacing them with robust heuristics.This shift is further quantified by the R^2 score, which measures the alignment between an agent's internal assessment and actual test performance. As shown by the corresponding Twitter alignment plots in the standalone supplementary appendix, the negative R^2 in the Base model (-5.02) reflects a severe semantic decoupling, where samples cluster in the "high validation, near-zero test" zone. EGAD restores the correlation ($R^2 = 0.75$) by shifting the distribution toward the $y = x$. This phenomenon suggests that even EDGA fails to eliminate the outlier, it does make the validation and test scores closer. Therefore, the visual and statistical results confirm that even without score feedback, entropy-guided diagnostics act as an endogenous filter, ensuring that validation gains represent genuine improvement rather than brittle over-fitting.

4.3.2 OPERATIONAL EFFICIENCY

Beyond accuracy, EGAD demonstrates superior computational efficiency. Across the *Twitter Sentiment*, *PetFinder*, and *Whale* competitions, our method reduces total execution time by 10% to 58% relative to the Base and Ablation models. This efficiency gain is a byproduct of improved diagnostic precision. In our three-stage pipeline (Generation, Revision, Debugging), traditional reflection often performs "blind" edits that fail to resolve underlying logical flaws, leading to redundant and time-consuming debugging loops—as evidenced by the extreme runtime (83,252s) of the Ablation model in the *Whale* task. By contrast, EGAD's entropy-guided "Rethink" mechanism localizes anomalies early in the revision stage. This targeted intervention increases the frequency of producing functionally correct code prior to execution, thereby preempting costly downstream debugging and accelerating the overall scientific discovery process.

In general, EGAD identifies high entropy at the point where the model attempts to bridge the gap between internal validation scores and the final test outcomes. By triggering a logic-centric "Rethink," EGAD successfully force the agent to give away the "cheat-to-win" strategy toward a robust, heuristic-based extractor. This results in a significant restoration of the correlation between validation and test performance, ensuring reliable autonomous evolution.

5 CONCLUSION

We presented EGAD, a training-free framework that enhances LLM code reasoning by monitoring token-level entropy aggregated to line-level and block-level means via AST decomposition. By detecting high-entropy lines as anomaly signals and triggering entropy-guided Rethink interventions, EGAD localizes and corrects logical flaws — including data leakage and algorithmic paradigm drift — without execution feedback or additional training. Experiments across HumanEval, LiveCodeBench, and MLE-bench on three model families demonstrate consistent accuracy gains over entropy-threshold-based parallel sampling and blind self-reflection baselines, with the most pronounced improvements on hard competitive programming problems and autonomous data science tasks.

Several directions remain open. First, the current anomaly detection relies on a fixed block-relative threshold; incorporating code complexity metrics or problem difficulty estimates could further improve precision. Second, while $\tau = 1.5$ works well across all tested models, designing a dynamic entropy threshold that adapts to problem difficulty and code length is a promising direction for improving robustness. Third, the entropy signal could serve as an intrinsic reward in reinforcement learning from verifiable rewards (RLVR), potentially internalizing uncertainty-awareness into model weights rather than applying it only at inference time.

AI USE STATEMENT

During preparation of this submission, generative AI tools were used to assist with LaTeX formatting and language editing. They were not used to generate experimental data or results. All AI-assisted work was reviewed, and the authors take responsibility for the final content, claims, and artifacts.

ETHICS STATEMENT

This work does not involve human subjects, sensitive personal data, or the release of new pretrained models. It studies inference-time improvements for code generation using existing public benchmarks and models.

REPRODUCIBILITY STATEMENT

The paper specifies the evaluated benchmarks, model families, baselines, metrics, and entropy threshold in Section 4. The code and additional implementation details will be released upon acceptance.

REFERENCES

Rui Ai, Yuqi Pan, David Simchi-Levi, Milind Tambe, and Haifeng Xu. Beyond majority voting: Llm aggregation by leveraging higher-order information. *arXiv preprint arXiv:2510.01499*, 2025.

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.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *URL https://arxiv.org/abs/2305.14325*, 3, 2023.

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yi Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and Hao Zhang. Efficiently scaling llm reasoning with certaindex. *arXiv preprint arXiv:2412.20993*, 2024.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence. *arXiv preprint arXiv:2508.15260*, 2025.

Baizhou Huang, Shuai Lu, Xiaojun Wan, and Nan Duan. Enhancing large language models in coding through multi-perspective self-consistency. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1429–1450, 2024.

Shiyu Ji, Yixuan Wang, Yijun Liu, Qingfu Zhu, and Wanxiang Che. Seer self-consistency: Advance budget estimation for adaptive test-time scaling. *arXiv preprint arXiv:2511.09345*, 2025.

Chen Jin, Ryutaro Tanno, Tom Diethe, and P. Teare. Corefine: Confidence-guided self-refinement for adaptive test-time compute. *arXiv preprint arXiv:2602.08948*, 2026.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*, 2023.

Hung Le, Hailin Chen, Amrita Saha, Akash Gokul, Doyen Sahoo, and Shafiq Joty. Codechain: Towards modular code generation through chain of self-revisions with representative sub-modules. *arXiv preprint arXiv:2310.08992*, 2023.

Yiwei Li, Ji Zhang, Shaoxiong Feng, Peiwen Yuan, Xinglin Wang, Jiayi Shi, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, et al. Revisiting self-consistency from dynamic distributional alignment perspective on answer aggregation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25208–25223, 2025.

Zhan Liu, Huaizhi Qu, Xiaowei Yin, He Sun, Yanjun Han, Tianlong Chen, and Zhun Deng. Pets: A principled framework towards optimal trajectory allocation for efficient test-time self-consistency. *arXiv preprint arXiv:2602.16745*, 2026.

Marcus J Min, Yangruibo Ding, Luca Buratti, Saurabh Pujar, Gail Kaiser, Suman Jana, and Baishakhi Ray. Beyond accuracy: Evaluating self-consistency of code large language models with identity-chain. *arXiv preprint arXiv:2310.14053*, 2023.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? *arXiv preprint arXiv:2306.09896*, 2023.

Daniel Scalena, Leonidas Zotos, Elisabetta Fersini, Malvina Nissim, and Ahmet Ustun. Eager: Entropy-aware generation for adaptive inference-time scaling. *arXiv preprint arXiv:2510.11170*, 2025.

Aman Sharma and Paras Chopra. The sequential edge: Inverse-entropy voting beats parallel self-consistency at matched compute. *arXiv preprint arXiv:2511.02309*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Ola Shorinwa, Zhiting Mei, Justin Lidard, Allen Z Ren, and Anirudha Majumdar. A survey on uncertainty quantification of large language models: Taxonomy, open research challenges, and future directions. *ACM Computing Surveys*, 58(3):1–38, 2025.

M. Vilas, Safoora Yousefi, Besmira Nushi, Eric Horvitz, and Vidhisha Balachandran. Tracing the traces: Latent temporal signals for efficient and accurate reasoning. *arXiv preprint arXiv:2510.10494*, 2025.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. Reasoning aware self-consistency: Leveraging reasoning paths for efficient llm sampling. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3613–3635, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Zenan Xu, Zexuan Qiu, Guanhua Huang, Kun Li, Siheng Li, Chenchen Zhang, Kejiao Li, Qi Yi, Yuhao Jiang, Bo Zhou, Fengzong Lian, and Zhanhui Kang. Adaptive termination for multi-round parallel reasoning: An universal semantic entropy-guided framework. *arXiv preprint arXiv:2507.06829*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Jinghan Zhang, Xiting Wang, Fengran Mo, Yeyang Zhou, Wanfu Gao, and Kunpeng Liu. Entropy-based exploration conduction for multi-step reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3895–3906, 2025.

Cheng Zhu, Siyang Wu, Xiangkang Zeng, Zishan Xu, Zhaolu Kang, Yifu Guo, Yuquan Lu, Jun Huang, and Guojing Zhou. Edis: Diagnosing llm reasoning via entropy dynamics. *arXiv preprint arXiv:2602.01288*, 2026.