

TrajAD: Trajectory Anomaly Detection for Trustworthy LLM Agents

Yibing Liu¹, Chong Zhang¹, Zhongyi Han^{1*}, Hansong Liu², Yong Wang², Yang Yu³, Xiaoyan Wang⁴ and Yilong Yin^{1†}

¹School of Software, Shandong University, Jinan, China

²Sonli Holding Group Co., Ltd., Qingdao, China

³Shandong Huazhi Talent Technology Co., Ltd., Jinan, China

⁴Information Technology Service Center of People’s Court

sduliuyb@163.com, zhangchongupc@163.com, hanzhongyicn@gmail.com, liuhansong@sonli.net, wangyong@sonli.net, yuy@sdas.org, 428163395@139.com, ylyin@sdu.edu.cn

Abstract

We address the problem of runtime trajectory anomaly detection, a critical capability for enabling trustworthy LLM agents. Current safety measures predominantly focus on static input/output filtering. However, we argue that ensuring LLM agents reliability requires auditing the intermediate execution process. In this work, we formulate the task of Trajectory Anomaly Detection. The goal is not merely detection, but precise error localization. This capability is essential for enabling efficient rollback-and-retry. To achieve this, we construct TrajBench, a dataset synthesized via a perturb-and-complete strategy to cover diverse procedural anomalies. Using this benchmark, we investigate the capability of models in process supervision. We observe that general-purpose LLMs, even with zero-shot prompting, struggle to identify and localize these anomalies. This reveals that generalized capabilities do not automatically translate to process reliability. To address this, we propose TrajAD, a specialized verifier trained with fine-grained process supervision. Our approach outperforms baselines, demonstrating that specialized supervision is essential for building trustworthy agents.

1 Introduction

LLM-based agents function as autonomous systems that leverage reasoning and planning to decompose complex goals into executable steps [Park *et al.*, 2023]. They have demonstrated potential in high-stakes domains, such as financial decision-making [Wang *et al.*, 2023; Zhou *et al.*, 2024] and clinical diagnosis [Singhal *et al.*, 2023; Tang *et al.*, 2024], where precision is paramount. However, despite this progress, the widespread deployment is hindered by safety concerns. In these safety-critical environments, the lack of robustness in the execution process poses severe risks.

*Corresponding author: Zhongyi Han, Email: hanzhongyicn@gmail.com

†Corresponding author: Yilong Yin, Email: ylyin@sdu.edu.cn

A critical challenge is the risk of trajectory anomalies. An agent’s execution involves complex interleaving of reasoning, tool usage, and environmental feedback. Due to this complexity, agents often commit errors in intermediate steps. Common anomalies include fabricating invalid tool parameters, entering infinite loops, or executing redundant actions that are locally plausible but globally inefficient. Crucially, these anomalies do not always result in immediate task failure. However, they lead to significant resource waste and potential safety risks, such as irreversible database corruption [Yuan *et al.*, 2024; Ying *et al.*, 2025]. This necessitates a mechanism to detect anomalies in the execution process and interrupt errors in real-time.

Current efforts primarily focus on enhancing capabilities or static safety, neither of which effectively addresses runtime trajectory anomalies. On the capability side, methods like trajectory-based fine-tuning [Chen *et al.*, 2023; Zeng *et al.*, 2024; Song *et al.*, 2024] and Process Reward Models (PRM) [Lightman *et al.*, 2023] introduce supervision during the training phase. However, they aim to optimize model parameters to improve the general policy. They do not function as a runtime monitor to audit specific execution instances. Similarly, safety measures such as hallucination detection [Zhang *et al.*, 2025b; Huang *et al.*, 2025] and safety guardrails [Zhang *et al.*, 2024; Dong *et al.*, 2025] operate on a local or static scope. They typically verify the final output against facts or filter atomic tool calls in isolation. Crucially, these methods lack temporal awareness. They fail to detect logical errors inherent to the sequence, such as infinite loops or redundant actions. Moreover, they cannot localize the specific error step. This necessitates a dedicated mechanism to verify the entire execution trajectory.

However, achieving this goal faces two primary obstacles. First, there is a lack of datasets that contrast normal trajectories with anomalous ones. Current datasets [Zeng *et al.*, 2024; Song *et al.*, 2024] rely on gold-standard trajectories to provide positive supervision. They rarely include annotated “negative samples” which are essential for learning to identify anomalies. Just as humans learn from mistakes, models require exposure to failure modes to establish robust decision boundaries. Second, precise anomaly detection and localization present significant challenges. Existing methods are optimized for instruction following and task completion. How-

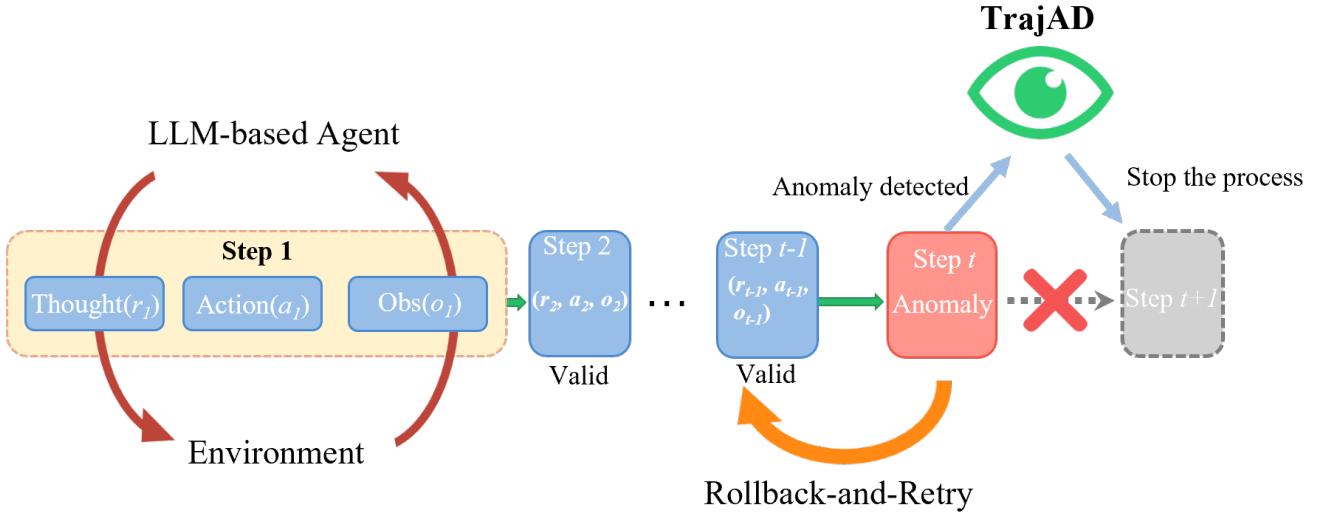


Figure 1: Overview of the TrajAD framework. We introduce the TrajAD framework to verify the agent’s trajectories. At each step, the agent generates a thought, takes an action, and receives an observation, forming an execution unit. The execution trajectory is periodically validated to check whether it remains normal. If all previous steps are valid, execution continues. When an anomaly is detected at step t , the process is halted before step $t+1$. The trajectory can rollback to the step $t-1$ and retry instead of restarting the whole task.

ever, they are not explicitly designed to differentiate between normal and anomalous behaviors. Furthermore, accurately pinpointing the exact position of an error is difficult. The boundary between a complex reasoning step and a redundant loop is often ambiguous without deep semantic understanding. Solving this localization problem significantly improves efficiency. Agents can “rollback” to the error step instead of restarting the entire task.

To address these challenges, we propose a systematic framework designed to proactively verify the execution process (Figure 1). We formally define the task of Trajectory Anomaly Detection. This task requires the model to distinguish anomalies based on global trajectory context. Crucially, it must also localize the exact error step to enable subsequent recovery. To achieve this, we construct TrajBench. We employ “Perturb-and-Complete” strategies on gold-standard trajectories to generate high-quality negative samples. We combine these with normal trajectories to form TrajBench. This dataset enables models to distinguish anomalies and locate errors. Using this dataset, we fine-tune Qwen3-4B to develop TrajAD as our core detector. Experiments show that general-purpose models struggle to distinguish anomalous trajectories. They fail even more on the challenging task of localizing exact error steps. In contrast, TrajAD achieves superior performance in both detection and localization.

Our main contributions are as follows:

- We identify and formalize the problem of Agent Trajectory Anomaly Detection. To the best of our knowledge, this is the first work to systematically investigate procedural anomalies in agent execution. We shift the evaluation paradigm from outcome-centric correctness to process-centric rationality, highlighting the critical need for runtime auditing mechanisms.

- We construct TrajBench, the first high-quality dataset dedicated to agent execution anomalies. It covers three representative anomaly categories: Task Failure, Process Inefficiency, and Unwarranted Continuation. To ensure generalization, we build upon AgentBank, which covers five core dimensions: reasoning, mathematics, coding, web navigation, and embodied AI. We modify these expert trajectories to synthesize corresponding fine-grained anomalies.

- We propose TrajAD, a specialized auditing framework for detecting and localizing anomalies. By modeling the global context of execution traces, TrajAD achieves precise step-level localization of errors. This capability enables efficient error recovery through a “rollback-and-retry” mechanism, significantly improving agent reliability and reducing resource consumption.

2 Related Work

2.1 Advancements in LLM-based Agents

Recent advancements in agentic systems primarily follow two paradigms based on architectural design and parameter update. Architectural design enhances LLM-based agents without modifying model weights. Reasoning frameworks decompose complex tasks into sequential thought processes [Wei *et al.*, 2022b; Yao *et al.*, 2023]. ReAct [Yao *et al.*, 2022] synergizes reasoning with acting by interleaving thoughts with observations. Memory mechanisms retrieve external knowledge [Lewis *et al.*, 2020] and past experiences [Zhang *et al.*, 2025a] to extend long-term memory. Furthermore, the Model Context Protocol (MCP)¹ standardizes tool integration, allowing agents to plug into dynamic

¹<https://modelcontextprotocol.io>

environments directly. Conversely, the parameter update paradigm embeds capabilities directly into the LLM’s inherent knowledge. General approaches align models with broad user intents via instruction tuning [Wei *et al.*, 2022a] and knowledge distillation [Hinton *et al.*, 2015]. Going further, Process Reward Models (PRMs) [Lightman *et al.*, 2023] introduce step-level supervision using human-labeled intermediate states. Trajectory-based fine-tuning [Chen *et al.*, 2023; Zeng *et al.*, 2024; Song *et al.*, 2024] optimize models on interaction trajectories. This enables agents to master the skills required for autonomous tasks.

However, these advancements prioritize capability over reliability. While agents can now handle harder problems, they exhibit “blind goal-directedness” [Shayegani *et al.*, 2025]. They greedily optimize for final outcomes while neglecting process rationality. This renders the execution process uncontrollable and risky. With increased autonomy, trajectory anomalies pose severe risks. Structural loops unnecessarily deplete computational budgets. More critically, unverified actions can trigger irreversible state changes, such as corrupting databases or executing unauthorized transactions. In safety-critical domains, such unstable behaviors undermine trust, making the agent unsafe regardless of the final outcome. Current methods focus on task success but lack the ability to verify their own intermediate steps.

2.2 Trustworthiness in Agents

Current research on agent trustworthiness primarily centers on hallucination detection, safety guardrails, and LLM-as-a-Judge. Hallucination detection targets factual correctness, checking textual consistency [Manakul *et al.*, 2023] or execution validity [Chern *et al.*, 2023] against ground truth. Safety guardrails deploy external filters [Inan *et al.*, 2023] or programmable rules [Rebedea *et al.*, 2023] to intercept adversarial attacks, preventing both toxic content generation, malicious prompts, and risky tool invocations [Yuan *et al.*, 2024; Ying *et al.*, 2025]. The “LLM-as-a-Judge” paradigm utilizes strong generalist models to grade the quality of generated content against human preferences [Zheng *et al.*, 2023; Bai *et al.*, 2022; Liu *et al.*, 2023].

However, these methods fail to monitor the dynamic execution process. Hallucination detection relies on static textual checks, while safety guardrails serve as passive defenses against external attacks. Similarly, current LLM judges rely on prompting for zero-shot evaluation. They lack the domain-specific knowledge to identify subtle anomalies within the process. In contrast, we synthesize anomaly trajectories to fine-tune a specialized verifier. This enables the model to grasp the logical dependencies between steps, allowing it to accurately identify anomalies and locate the exact step.

3 Problem Formulation

In this section, we establish the formal framework for Trajectory Anomaly Detection. We first model the agent execution as a sequential decision process. Unlike outcome-centric evaluations, we characterize anomalies based on *process rationality*, focusing on three primary categories of anomalies. Finally, we formulate the auditing task as a supervised learn-

ing problem, where the objective is to jointly predict the anomaly verdict and localize the specific error step.

3.1 Preliminaries

We formulate agent execution as a sequential decision process. Given a task instruction I , the agent interacts with the environment over n steps. At each step t , the agent first generates a thought r_t for planning. Conditioned on this reasoning, it executes an action a_t . Upon receiving the action, the environment transitions to a new state and returns an observation o_t reflecting this change. This interaction cycle repeats until the task is completed. We define the interaction trajectory T as the sequence of these triplets:

$$T = \{I, (r_1, a_1, o_1), (r_2, a_2, o_2), \dots, (r_n, a_n, o_n)\}, \quad (1)$$

where n denotes the total number of steps. This formulation explicitly captures the interleaving of reasoning, execution, and feedback.

3.2 Taxonomy of Anomalies

Unlike outcome-based evaluation, we focus on the rationality of the execution process. We focus on three primary categories of anomalies:

- **Type I: Task Failure (\mathcal{A}_{fail})**. The agent fails to complete the task. This includes two cases:
 - (a) *Reasoning Error*: The agent executes a valid action a_t based on flawed reasoning r_t .
 - (b) *Execution Error*: The agent executes an incorrect action a_t , causing runtime exceptions.
- **Type II: Process Inefficiency (\mathcal{A}_{ineff})**. The agent completes the task, but with redundant steps. Formally, a trajectory is inefficient if a shorter trajectory T' exists that achieves the same outcome (i.e., $|T'| < |T|$). This includes circular loops or extra actions that are locally plausible but globally inefficient.
- **Type III: Unwarranted Continuation (\mathcal{A}_{unw})**. The agent fails to stop when tasks are impossible or unnecessary due to environment changes.
 - (a) *Failure to Refuse*: The task is impossible under current constraints. The agent fails to report the inability and hallucinates a plan.
 - (b) *Redundant Continuation*: The task is already finished or the context has changed, making further actions meaningless. The agent fails to perceive this termination condition and continues execution.

3.3 Task Definition

We define Trajectory Anomaly Detection as a supervised auditing task. Given a trajectory T , the goal is to learn a mapping function $f : T \rightarrow (c, l)$.

Here, $c \in \{\text{Normal, Anomaly}\}$ represents the binary verdict of the trajectory’s validity. The variable l denotes the *First Error Step*:

$$l = \begin{cases} t_{err}, & \text{if } c = \text{Anomaly}; \\ \emptyset, & \text{if } c = \text{Normal}, \end{cases} \quad (2)$$

where $t_{err} \in \{1, \dots, n\}$ is the index of the first step where the anomaly occurs. Precise prediction of t is critical, as it enables the agent to rollback to the pre-error state s_{t-1} for efficient recovery, rather than restarting the entire task.

4 TrajBench: A Dataset for Trajectory Anomaly Detection

To enable the auditing task defined in Sec. 3, a dataset containing both execution anomalies and precise error localization is required. Existing benchmarks [Song *et al.*, 2024] primarily target imitation learning, consisting solely of expert demonstrations. They lack the negative samples and step-wise annotations necessary for learning process verification. To bridge this gap, we construct TrajBench, a large-scale dataset synthesized via a semi-automated pipeline. TrajBench explicitly pairs golden trajectories with strictly defined anomalies, providing full supervision for both the anomaly verdict c and error location t .

4.1 Data Construction Pipeline

We utilize AgentBank [Song *et al.*, 2024] as our seed dataset due to its broad coverage across five core domains: Reasoning, Math, Programming, Web Navigation, and Embodied AI. To ensure the quality of the base data, we first employ a validator model to filter the raw trajectories, retaining only those with logically sound reasoning chains. Based on these verified seeds, we apply a Perturb-and-Complete strategy to synthesize negative samples. This process involves three steps:

Step 1: Perturbation Injection. Given a golden trajectory T_{gold} , we sample a target step t . To ensure the verifier captures global context, we prioritize sampling from intermediate positions rather than early steps. We inject a perturbation into step t to create a deviated state, strictly following the anomaly taxonomy in Sec. 3.2:

- **Type I: Task Failure (\mathcal{A}_{fail})**. We inject fatal errors into the execution stream. For *Reasoning Errors*, we replace the valid thought r_t with a logical flaw or state misconception. For *Execution Errors*, we modify action a_t to invoke incorrect tools or invalid parameters.
- **Type II: Process Inefficiency (\mathcal{A}_{ineff})**. We introduce redundancy without altering the final outcome. We insert semantically valid but useless sub-sequences (e.g., loops $A \rightarrow B \rightarrow A$ or detours $A \rightarrow C \rightarrow B$) into the trajectory. These actions appear locally plausible but waste computational resources, challenging the verifier to identify global inefficiency.
- **Type III: Unwarranted Continuation (\mathcal{A}_{unw})**. We manipulate the termination conditions. To simulate *Failure to Refuse*, we remove necessary tools or set conflicting constraints, forcing the agent to hallucinate a plan. To simulate *Redundant Continuation*, we inject a “Task Completed” signal into observation o_t but instruct the agent to ignore it and continue execution.

Step 2: Conditional Completion. Conditioned on the perturbed history $T_{\leq t}$, a strong language model generator is employed to simulate the subsequent behavior $T_{>t}$. We explicitly constrain the generation to maintain logical consistency

with the injected error (e.g., continuing a wrong path after a reasoning error) to ensure the trajectory remains coherent.

Step 3: Automatic Annotation. A key advantage of this pipeline is the acquisition of precise labels without manual cost. Since the perturbation step t is controlled, we automatically assign the ground truth error location $L_{loc} = t$ and the verdict $C_{verdict} = \text{Anomaly}$. The valid seed trajectory serves as the positive sample ($C_{verdict} = \text{Normal}$).

4.2 Dataset Statistics and Quality Analysis

TrajBench comprises a total of 60,000+ trajectories, strictly balanced with a 1:1 ratio between normal and anomalous samples. The dataset covers 13 tasks across five domains, ensuring broad diversity. The anomalies are evenly distributed, with Type I, II, and III accounting for roughly 33% each (see Section 1 of the Supplementary Material for details).

To ensure high quality, we implement a rigorous two-stage verification process. During the seed collection phase, we employed a validator model to verify the logical consistency of the source trajectories. Only samples with coherent reasoning chains were retained, resulting in a pass rate of 91.6% (retaining 34436 out of 37625 raw seeds). This ensures that our positive samples are strictly “golden”. From these verified seeds, our pipeline successfully synthesized 31,742 valid anomalous trajectories (a generation success rate of 92.2%). The remaining failures were due to model refusal or format parsing errors. To establish a rigorous evaluation setting, we construct TrajBench by pairing each successfully synthesized anomaly with its original source trajectory. This results in a strictly balanced dataset of 63,484 samples, eliminating class distribution bias.

Finally, to assess the reliability of the synthesized labels, we conduct a human review on a stratified random subset of 500 samples (100 per domain). Annotators verify two criteria: (1) anomaly category alignment, and (2) error step localization precision. The results show a Human-Model Agreement rate of 96.2% for classification and 94.5% for localization. This high consistency confirms that our automated pipeline produces trusted supervision signals.

5 TrajAD: A Generative Verifier for Agent Trajectories

We propose TrajAD, a generative verifier designed for the auditing task defined in Sec. 3. We formulate the problem as conditional text generation and fine-tune the model on the TrajBench dataset (Sec. 4). Formally, the input sequence \mathcal{X} consists of a system instruction I_{sys} and the trajectory T . Given an input sequence $\mathcal{X} = \{I_{sys}, T\}$, the model generates a structured diagnostic report \mathcal{Y} :

$$\mathcal{Y} = [C_{cls}; L_{loc}], \quad (3)$$

where $C_{cls} \in \{\text{Normal}, \text{Anomaly}\}$ denotes the verdict and $L_{loc} \in \{1, \dots, n\}$ denotes the index of the error step.

We adopt a standard decoder-only Transformer as the backbone. To adapt the model to the auditing task while maintaining computational efficiency, we employ Low-Rank Adaptation (LoRA) [Hu *et al.*, 2022]. We freeze the pre-trained weights W_0 and introduce trainable low-rank matrices A and

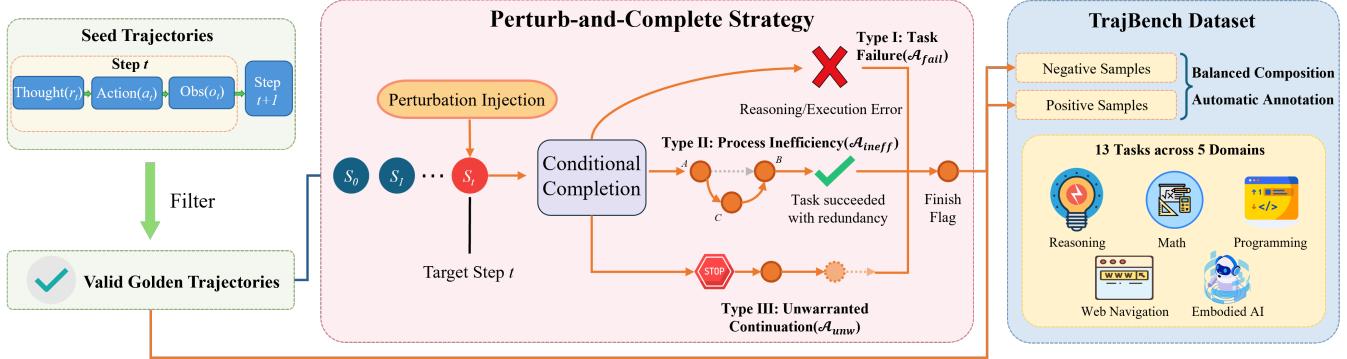


Figure 2: The data construction pipeline for TrajBench. We initialize the process by filtering seed trajectories to ensure a high-quality set of valid golden trajectories. To construct negative samples, we employ a Perturb-and-Complete strategy. We inject a perturbation into a target step S_t and force conditional completion to finish the subsequent trajectory based on the altered context. This process synthesizes three distinct anomaly types: Task Failure (\mathcal{A}_{fail}), Process Inefficiency (\mathcal{A}_{ineff}), and Unwarranted Continuation (\mathcal{A}_{unw}). The resulting dataset features a balanced composition of positive and negative samples, where anomalous trajectories are automatically annotated. The dataset consists of 13 tasks across 5 domains.

B. The forward pass is modulated as $h = (W_0 + BA)x$. We optimize the parameters $\Phi = \{A, B\}$ using the standard autoregressive objective over the output \mathcal{Y} in TrajBench:

$$\mathcal{L} = - \sum_{t=1}^{|\mathcal{Y}|} \log P(y_t | \mathcal{X}, y_{<t}), \quad (4)$$

This formulation allows the model to learn the joint distribution of anomaly verdicts and error locations directly from the supervision signals provided in TrajBench.

During inference, TrajAD operates as a runtime monitor embedded in the agent’s execution loop (Figure 1). We perform verification at a fixed step interval. The model takes the current trajectory history as input and predicts the tuple (C_{cls}, L_{loc}) . The inference process follows a “Check-and-Act” protocol: If $C_{cls} = \text{Normal}$, the agent continues execution. If $C_{cls} = \text{Anomaly}$, the execution is interrupted. The agent then utilizes the predicted index in L_{loc} to rollback the environment to the pre-error state s_{l-1} , enabling targeted recovery without a full restart.

6 Experiments

We evaluate TrajAD on the TrajBench dataset to validate its effectiveness in verifying agent trajectories. Our experiments focus on three key questions: (1) Does specialized trajectory detection outperform general-purpose reasoning? (2) Can the model robustly localize errors across diverse domains? (3) How does data scale impact the verification capability?

6.1 Experimental Setup

Dataset and Baselines. We utilize the balanced TrajBench dataset (60k samples) constructed in Sec. 4. We adopt a stratified split, reserving 10% of samples from each task for testing. We benchmark TrajAD (fine-tuned Qwen3-4B) against representative zero-shot baselines. We select models to evaluate distinct hypotheses:

- **Qwen3-4B (Base) [Yang et al., 2025] & Gemma-3-4B-Instruct [Kamath et al., 2025]:** As general-purpose

models of the same scale, they serve to evaluate whether standard instruction-following capabilities are sufficient for anomaly auditing without specialized supervision.

- **Phi-3-Mini-4k-Instruct [Abdin et al., 2024]:** We include this lightweight model to benchmark the reasoning capabilities of small-scale LLMs across the full dataset.
- **Qwen3-8B [Yang et al., 2025]:** We include a larger-scale model to investigate whether simply scaling model capacity can solve the auditing challenge without specific fine-tuning.

Evaluation Metrics. We employ a multi-dimensional evaluation suite (see the Supplementary Material for complete definitions and details):

- **Detection (Binary Classification):** We report Precision (\mathcal{P}), Recall (\mathcal{R}), and Macro-F1 (\mathcal{F}_1). We prioritize Recall to minimize safety risks associated with missed anomalies.
- **Localization (Joint Verification):** We define a strict **Joint Exact Match (JEM)** metric. A prediction is considered correct if and only if:

1. The predicted error step index l_{pred} exactly matches the ground truth l_{gt} .
2. The semantic similarity between the generated error content c_{pred} and the ground truth c_{gt} exceeds a threshold τ .

Formally, $JEM = \mathbb{I}(l_{pred} = l_{gt}) \cdot \mathbb{I}(\text{sim}(c_{pred}, c_{gt}) > \tau)$. We compute similarity using the Ratcliff-Obershelp algorithm (via Python’s `difflib` module). We set $\tau = 0.2$ to allow for diverse phrasing while rejecting irrelevant content. Our preliminary verification experiments reveal that models can achieve artificially high localization scores (e.g., $\sim 71.5\%$). A closer inspection of the outputs indicates that models often guess the correct index without identifying the actual anomaly location. JEM evaluates whether the model correctly identifies *why* a step is anomalous, rather than simply guessing *where* it is.

Table 1: Main Results: The overall performance comparison of TrajAD against baseline models. We report Precision (\mathcal{P}), Recall (\mathcal{R}), and Macro-F1 (\mathcal{F}_1) for anomaly detection, and Joint Exact Match (JEM) for error step localization. The best results are highlighted in **bold**, and the second-best results are underlined.

Model	Params	Method	Anomaly Detection(%)			Localization(%)
			Precision(\mathcal{P})	Recall(\mathcal{R})	Macro-F1(\mathcal{F}_1)	
Open-Source Baselines						
Gemma-3-4B-Instruct	4B	Zero-shot	68.64	64.66	64.20	<u>9.07</u>
Phi-3-Mini	4B	Zero-shot	67.78	28.46	30.65	3.28
Qwen3-4B	4B	Zero-shot	<u>79.07</u>	68.97	<u>70.43</u>	5.54
Qwen3-8B	8B	Zero-shot	76.16	<u>69.60</u>	67.90	5.81
Ours						
TrajAD(Ours)	4B	LoRA finetune	82.90	82.49	81.81	53.75

Case Study: Redundancy loops in an Embodied AI Task	
Task Definition: Instruction : “Clean a plate and put it in the cabinet.”	
Trajectory Snippet (Compressed):	
<ul style="list-style-type: none"> [Step 01–06] Navigate to Sink & PickUp(Plate) & Put(Plate, Sink) • [Step 07] ToggleObject (Faucet) ← Action: Stop Cleaning (State Changed: Cleaned) ⚠ [Step 08] ToggleObject (Faucet) ← The Anomaly: Redundant Cleaning ⚠ [Step 09] ToggleObject (Faucet) ← The Anomaly: Redundant Cleaning • [Step 10–16] Navigate to Cabinet & Put(Plate, Cabinet) (Task Completed) 	
Baseline Model: Prediction: Normal <i>“The agent cleaned the plate successfully ...”</i>	TrajAD (Ours): Prediction: Anomaly <i>“Anomaly at Step 8. The plate state is already ‘Cleaned’ after Step 7...”</i>

Figure 3: Qualitative Comparison on Redundancy Loops. The baseline model overlooks the repeated cleaning action since it does not affect the final goal state. In contrast, TrajAD detects the redundancy as a inefficiency process.

Implementation Details. We fine-tune the Qwen3-4B base model using QLoRA [Dettmers *et al.*, 2023]. We attach Low-Rank Adapters ($r = 8, \alpha = 16$) to all linear layers, affecting only 1.8% of total parameters. Training employs the Paged AdamW 8-bit optimizer with a peak learning rate of 2×10^{-5} and a 10% warmup. All experiments are conducted on a single NVIDIA A100 (80GB) GPU.

6.2 Main Results

We first evaluate the model’s performance under the In-Distribution (ID) setting, where the training and testing samples are drawn from the same set of 13 tasks across 5 do-

mains. Note that while the tasks are seen, the specific test trajectories are strictly held-out.

Table 1 summarizes the performance. TrajAD achieves a substantial improvement over all baselines, validating the effectiveness of trajectory anomaly detection. As shown in Table 1, zero-shot models exhibit a critical Precision-Recall imbalance. For instance, Qwen3-4B achieves a high Precision of 79.07% but a low Recall of 68.97%, while Phi-3 suffers from a severe Recall collapse (28.46%). This indicates a conservative bias. Pre-trained models tend to assume agent actions are valid, failing to detect subtle anomalies. This leads to a high false-negative rate. Furthermore, their localization capability is virtually non-existent, with Joint Exact Match (JEM) scores consistently below 10%. This indicates that general-purpose LLMs lack the capability to precisely localize errors within long trajectory sequences. As shown in Figure 3, baselines often overlook subtle procedural anomalies (e.g., redundant actions) as long as the final goal is achieved.

In contrast, TrajAD effectively overcomes these limitations. It improves Macro-F1 by 11.38% (to 81.81%) compared to the strongest baseline. More importantly, it achieves a breakthrough in localization, boosting JEM by 48.21% (to 53.75%). This demonstrates that our generative objective successfully forces the model to couple logical reasoning with structural verification, enabling precise error diagnosis.

To analyze performance stability, we decompose the ID evaluation into five domains: Math, Reasoning, Coding, Web Navigation, and Embodied AI. Figure 4 illustrates the results on each domain. TrajAD consistently outperforms baselines across all five domains. Zero-shot baselines fail to ground actions in Embodied AI tasks, resulting in near-zero localization. In contrast, TrajAD maintains high precision in this complex domain. This confirms that under the ID setting, our method successfully masters the distinct verification logic required for each domain.

6.3 Out-of-Distribution Generalization

A critical question is whether TrajAD learns universal verification logic or simply memorizes domain-specific patterns. To investigate this, we conduct a Cross-Domain Transfer experiment under the Out-of-Distribution (OOD) setting.

To ensure the target task is unseen, we adopt a strict Leave-One-Domain-Out protocol. We select Embodied AI

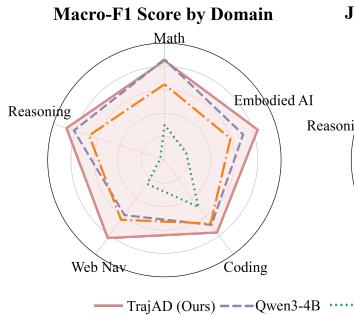


Figure 4: Domain-Specific Performance Analysis. (Left) Macro-F1 across five domains. TrajAD (solid red) forms the outermost envelope, demonstrating consistent robustness. (Right) Exact Match. Baselines exhibit a structural collapse near the center, highlighting their inability to localize errors, whereas TrajAD maintains a functional verification boundary.

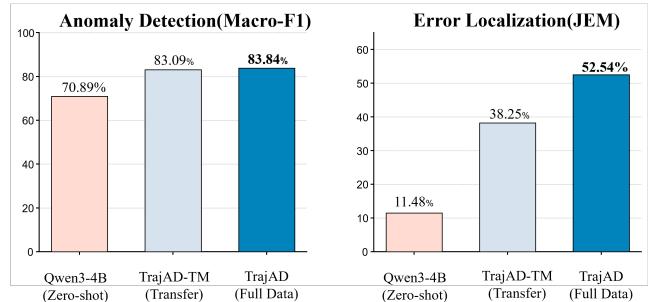
as the held-out target domain D_{target} and train a transfer model, TrajAD-TM, on the remaining source domains $D_{source} = \{\text{Math, Reasoning, Coding, Web}\}$. We then evaluate this model directly on D_{target} without any further adaptation. This rigorous setting tests the model’s ability to transfer verification logic to a novel action space without relying on memorized domain patterns.

As shown in Figure 5a, TrajAD-TM exhibits strong transferability, outperforming the zero-shot baseline on the unseen domain. Specifically, it improves Macro-F1 from 70.89% to 83.09% and JEM from 11.48% to 38.25%. However, a performance gap remains when compared to the fully supervised upper bound. While detection performance is nearly identical (83.09% vs. 83.84% F1), localization still lags behind the supervised model (38.25% vs. 52.54% JEM). This indicates that localization is more sensitive to subtle variations in anomaly patterns across domains. This sensitivity suggests that TrajAD can serve as a probe to extract domain-specific failure modes, facilitating targeted improvements in agents.

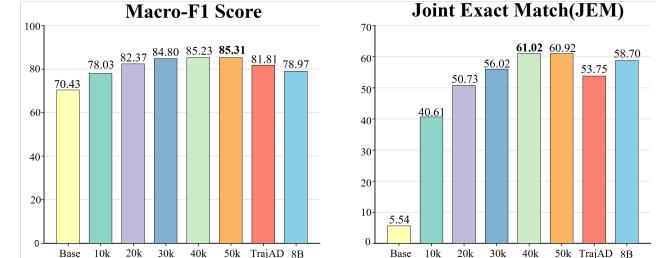
6.4 Scaling and Efficiency Analysis

Finally, we investigate the efficiency of our framework by analyzing the impact of training data scale and model capacity. We fine-tune TrajAD on stratified subsets ranging from 10k to 60k samples. As illustrated in Fig. 5b, performance correlates positively with data size in the early stages. Increasing samples from 10k to 50k yields consistent improvement, peaking at 85.31% F1 and 61.02% JEM.

To investigate model capacity constraints, we extend our evaluation to the larger Qwen3-8B model. First, in the zero-shot setting (Table 1), the 8B base model fails to outperform the 4B base model (67.90% vs. 70.43% F1), suggesting that raw parameter count alone does not confer an advantage in auditing logic. We further fine-tune the 8B model on the full dataset. As shown in Fig. 5b, this larger model achieves 78.97% F1. Notably, this does not surpass the TrajAD model trained on the same data partition (81.81% F1), nor does it reach the optimal 4B checkpoint (85.31% F1). Even with



(a) Generalization Capabilities. TrajAD (Transfer) demonstrates strong zero-shot detection performance on the held-out Embodied AI domain, though localization benefits from in-domain training.



(b) Scaling Law & Model Capacity. Performance improves linearly with data scale up to 50k samples. Scaling to 60k introduces negative transfer, and increasing model size (8B) does not overcome this distribution bottleneck.

Figure 5: Ablation and Analysis Experiments. (a) Generalization: Comparison of zero-shot transfer (Transfer Model) vs. full supervision. The detection gap is minimal, validating the universality of the learned logic. (b) Scalability: Impact of training data size and model parameters. The 4B model with 50k stratified samples achieves optimal efficiency, outperforming both the full-data 4B model and the larger 8B baseline.

fine-tuning, scaling up the model yields limited gains. This suggests that parameter scale is not the primary bottleneck for this task. Consequently, the 4B model demonstrates a superior trade-off between performance and computational cost, making it the more efficient choice for deployment.

7 Conclusion

In this work, we have addressed the challenge of ensuring agent reliability by formally defining the task of Trajectory Anomaly Detection. We construct TrajBench, the first large-scale benchmark dedicated to this purpose. Our experiments show that general-purpose LLMs fail to localize errors, regardless of model scale. They lack the capability to link detected anomalies to specific execution steps. We show that scaling up model size does not solve this issue. Specialized supervision is necessary. Our method, TrajAD, outperforms larger baselines. This proves that a small model is effective when trained to explicitly generate the error details. We hope this work serves as a foundational step, shifting the focus of agent evaluation from outcome-based metrics to rigorous process auditing, ultimately paving the way for trustworthy autonomous systems with minimal human intervention.

References

- [Abdin *et al.*, 2024] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.
- [Bai *et al.*, 2022] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- [Chen *et al.*, 2023] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.
- [Chern *et al.*, 2023] I Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, Pengfei Liu, et al. Factool: Factuality detection in generative ai—a tool augmented framework for multi-task and multi-domain scenarios. *arXiv preprint arXiv:2307.13528*, 2023.
- [Dettmers *et al.*, 2023] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient fine-tuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.
- [Dong *et al.*, 2025] Yi Dong, Ronghui Mu, Yanghao Zhang, Sisi Sun, Tianle Zhang, Changshun Wu, Gaojie Jin, Yi Qi, Jinwei Hu, Jie Meng, et al. Safeguarding large language models: A survey. *Artificial intelligence review*, 58(12):382, 2025.
- [Hinton *et al.*, 2015] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [Hu *et al.*, 2022] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- [Huang *et al.*, 2025] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.
- [Inan *et al.*, 2023] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.
- [Kamath *et al.*, 2025] Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, et al. Gemma 3 technical report. *CoRR*, 2025.
- [Lewis *et al.*, 2020] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.
- [Lightman *et al.*, 2023] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.
- [Liu *et al.*, 2023] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using gpt-4 with better human alignment. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore, December 2023. Association for Computational Linguistics.
- [Manakul *et al.*, 2023] Potsawee Manakul, Adian Liusie, and Mark Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 9004–9017, 2023.
- [Park *et al.*, 2023] Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.
- [Rebedea *et al.*, 2023] Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. In *Proceedings of the 2023 conference on empirical methods in natural language processing: system demonstrations*, pages 431–445, 2023.
- [Shayegani *et al.*, 2025] Erfan Shayegani, Keegan Hines, Yue Dong, Nael Abu-Ghazaleh, Roman Lutz, Spencer Whitehead, Vidhisha Balachandran, Besmira Nushi, and Vibhav Vineet. Just do it! computer-use agents exhibit blind goal-directedness. *arXiv preprint arXiv:2510.01670*, 2025.
- [Singhal *et al.*, 2023] Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.
- [Song *et al.*, 2024] Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2124–2141, Mi-

- ami, Florida, USA, November 2024. Association for Computational Linguistics.
- [Tang *et al.*, 2024] Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 599–621, 2024.
- [Wang *et al.*, 2023] Neng Wang, Hongyang Yang, and Christina Dan Wang. Fingpt: Instruction tuning benchmark for open-source large language models in financial datasets. *arXiv preprint arXiv:2310.04793*, 2023.
- [Wei *et al.*, 2022a] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.
- [Wei *et al.*, 2022b] 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.
- [Yang *et al.*, 2025] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [Yao *et al.*, 2022] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.
- [Yao *et al.*, 2023] 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.
- [Ying *et al.*, 2025] Zonghao Ying, Yangguang Shao, Jianle Gan, Gan Xu, Junjie Shen, Wenxin Zhang, Quanchen Zou, Junzheng Shi, Zhenfei Yin, Mingchuan Zhang, et al. Securewebarena: A holistic security evaluation benchmark for l1lms-based web agents. *arXiv preprint arXiv:2510.10073*, 2025.
- [Yuan *et al.*, 2024] Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, et al. R-judge: Benchmarking safety risk awareness for llm agents. *arXiv preprint arXiv:2401.10019*, 2024.
- [Zeng *et al.*, 2024] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, 2024.
- [Zhang *et al.*, 2024] Zhixin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024.
- [Zhang *et al.*, 2025a] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanduru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.
- [Zhang *et al.*, 2025b] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren’s song in the ai ocean: A survey on hallucination in large language models. *Computational Linguistics*, pages 1–46, 2025.
- [Zheng *et al.*, 2023] Liannmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.
- [Zhou *et al.*, 2024] Tianyu Zhou, Pingqiao Wang, Yilin Wu, and Hongyang Yang. Finrobot: AI agent for equity research and valuation with large language models. In *ICAIIF 2024: The 1st Workshop on Large Language Models and Generative AI for Finance*, 2024.