CERTAGENT: CONFORMAL RISK CONTROL FOR AGENT RUNTIME FRAGILITY CERTIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents extend language models from text generation to interactive task solving, but their deployment can fail through fragile runtime behaviors arising from reasoning or action errors. Existing mitigation methods improve empirical robustness, but they do not provide formal risk guarantees over the entire Agent Runtime. In this paper, we propose *CertAgent*, a framework for Agent Runtime Fragility Certification. We first formulate *Agent Runtime Fragility* as system level risk over the full agent runtime trajectory, while using stage specific failure modes to guide runtime control. CertAgent instantiates a Verifier Subagent to detect unreliable runtime proposals, a Gate Module to control runtime conservativeness through a configuration parameter λ, and a reflection mechanism to revise rejected proposals. To provide deployment time guarantees, we apply Conformal Risk Control to certify Agent Runtime Fragility, enabling users to either obtain a certified runtime risk bound α for a fixed configuration or identify certified runtime configurations under a target risk level. We further introduce policy optimization to reduce system level runtime risk and improve the resulting certification. Experiments on representative agent benchmarks show that CertAgent improves runtime reliability, supports practical risk utility tradeoffs, and yields more effective certification. Our code is available at https://anonymous.4open.science/r/CertAgent.

1 INTRODUCTION

LLM agents (Wang et al., 2024; Xi et al., 2025; Huang et al., 2024; Mohammadi et al., 2025; Acharya et al., 2025; Shavit et al., 2023; Schneider, 2025; Murugesan, 2025; Hosseini & Seilani, 2025) have emerged as a powerful paradigm for extending language models from text generation to interactive task solving. By combining multi step reasoning with external actions, LLM agents can decompose complex goals, invoke tools, interact with environments, update their decisions from feedback, and complete tasks that are difficult for language models alone. Recent systems such as coding agents (Tang et al., 2024; Zhang et al., 2024; Robeyns et al., 2025; Trivedi et al., 2024), browser agents (Hong et al., 2024; You et al., 2024; Zheng et al., 2024), and scientific agents (Ren et al., 2025; Wang et al., 2022; Ghafarollahi & Buehler, 2025; Lála et al., 2023; Gridach et al., 2025) have demonstrated strong potential across software engineering (Liu et al., 2024; He et al., 2025; Yang et al., 2024; Da et al., 2025), web navigation (Chae et al., 2024; He et al., 2024; Ning et al., 2025), data analysis (Hong et al., 2025; Goswami et al., 2025), and general assistance (Jia et al., 2025; Kang et al., 2025). These capabilities can substantially improve productivity, automate complex workflows, and enable new forms of human AI collaboration.

Nevertheless, Agent Runtime can still fail during deployment. Such failures may be unacceptable in high stakes scenarios such as healthcare (Moritz et al., 2025; Karunanayake, 2025; Wang et al., 2025b), finance (Axtell & Farmer, 2025; Li et al., 2025), and cybersecurity (Wang et al., 2025c; Landolt et al., 2025; Saha & Shukla, 2025). Recent studies have shown that LLM agents can be fragile during runtime, where errors may arise from flawed reasoning, inappropriate tool use, incomplete action arguments, execution failures, or misleading intermediate observations (Barke et al., 2026; Zhu et al., 2025; Zhang et al., 2025; Zhao et al., 2024; Song et al., 2024; Pan et al., 2025; Ruan et al., 2023). Existing methods such as self reflection (Shinn et al., 2023; Renze & Guven, 2024), tool use validation (Wang et al., 2025a), and runtime monitoring (Koohestani, 2025; Bhardwaj, 2026) attempt to mitigate these failures and improve empirical robustness. However, they generally do not provide a formal risk guarantee over the entire Agent Runtime. Such a guarantee is critical for deployment

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.

because users need to know not only whether an agent performs well empirically, but also whether its runtime risk can be controlled under a specified tolerance. To the best of our knowledge, formal certification for controlling system level Agent Runtime fragility remains largely unexplored.

Establishing such certification remains a substantial challenge. First, runtime failures are heterogeneous and stage dependent. A reasoning failure may come from incorrect control or insufficient reasoning support, while an action failure may come from invalid tool selection, missing arguments, or ineffective execution. A certification framework should evaluate stage specific failure modes and enable reflection when a runtime proposal is judged unreliable. Second, the standard calibration tools do not apply out of the box. Conformal risk control requires the runtime loss to be non increasing in the conservativeness parameter and to already meet the target at the strictest configuration. Neither is a property of an Agent Runtime: a stricter gate suppresses unreliable proposals but also triggers more reflection, and an execution that exhausts its reflection budget is a failure rather than a safe outcome. Third, a valid certificate is not automatically an actionable one. It must state what a user obtains from the single calibration set in hand, must remain valid for the configuration the user deploys rather than only for the one a selection rule returns, and must be able to report that a target level is not attainable instead of silently returning the most conservative candidate. Together, these challenges make effective certification over the entire Agent Runtime nontrivial.

In this work, we propose *CertAgent*, a framework for Agent Runtime Fragility Risk Control. We first introduce *Agent Runtime Fragility* to characterize the vulnerability of an Agent Runtime to failures that emerge during sequential reasoning and action. To address heterogeneous runtime failures, CertAgent uses a stage structured ReAct runtime (Yao et al., 2022), a widely adopted paradigm for describing agent execution. This paradigm allows the Verifier Subagent to be instantiated separately for the Reasoning Stage and the Action Stage, enabling it to score proposal reliability and identify stage specific failure modes. To support different deployment risk preferences, CertAgent introduces a shared Gate Module controlled by a runtime configuration parameter λ, which decides whether a proposal should be allowed or revised through reflection. To certify this runtime, we establish the two conditions required by calibration rather than assume them. A pathwise argument shows that raising the acceptance threshold causes the runtime to deploy a proposal whose verifier score is at least as large, and a non increasing majorant then enforces the condition by construction at a cost we quantify on the calibration set; an explicit abstention option makes the target attainable at every level and turns an infeasible requirement into a reported outcome. We then give an expectation certificate valid for every configuration in the certified set, so that deployment may select inside it by utility, together with an exact finite sample certificate that follows from exchangeability alone and under which the two deployment views are one order statistic read along two axes. Finally, we introduce policy optimization to reduce system level runtime risk, thereby improving the resulting certification and enabling less conservative certified configurations. Our contributions are summarized as follows:

- **Problem Formulation.** We introduce *Agent Runtime Fragility* to characterize reliability failures that may occur in both reasoning and action stages. We identify stage specific failure modes, formulate the system level risk over the *entire* Agent Runtime as the certification target, and charge the maximal loss to abandoned executions so that conservativeness is not rewarded by construction.
- **Novel Certification Methodology.** We propose *CertAgent*, a stage structured framework in which the Verifier Subagent detects failures, the Gate Module controls conservativeness, and reflection revises proposals. We establish conditions for distribution free calibration through pathwise monotonicity, a non increasing majorant, and explicit abstention. This yields an expectation guarantee across the certified set and an exact finite sample guarantee unifying deployment views.
- **Empirical Evaluation.** We evaluate CertAgent on representative agent benchmarks and show that it improves runtime reliability while supporting risk utility tradeoffs. We report the diagnostics that make the calibration conditions auditable, and show that policy optimization reduces system level runtime risk, yielding tighter certified bounds and an enlarged certified configuration set.

2 PRELIMINARY

2.1 NOTATIONS

We consider a ReAct (Yao et al., 2022) agent over a controlled runtime trajectory $\xi(\lambda) = (s_1, \ldots, s_T)$, where $s_t \in \mathcal{S}$ is the runtime state and $\lambda \in \Lambda$ is the global runtime configuration. Configurations are

selected from a finite grid $\Lambda = \{\lambda_1 < \cdots < \lambda_K\} \subset [0,1]$ fixed before calibration, since deployment selects a single operating point from a small candidate set. The runtime has two stages $m \in \{R, A\}$, with proposal $z_t^m \in \mathcal{Z}_m$, where $\mathcal{Z}_R = \mathcal{R}$ and $\mathcal{Z}_A = \mathcal{A}$. Equivalently, $z_t^R = r_t$ is a reasoning proposal and $z_t^A = a_t$ is an action proposal. Each stage has a verifier V_m. Given (s_t, z_t^m), the verifier returns a score $q_t^m \in [0,1]$ and feedback $e_t^m \in \mathcal{E}$, where $\mathcal{E}$ is the space of free form textual feedback for reflection. A non decreasing map τ converts the configuration into an acceptance threshold, and the shared gate is $G(q_t^m; \lambda) = \mathbf{1}\{q_t^m \geq \tau(\lambda)\}$, where 1 means allow and 0 means reflect; larger λ denotes stricter acceptance. The stage specific reflection operator $\mathrm{Reflect}_m$ revises rejected proposals in $\mathcal{Z}_m$ using feedback from $\mathcal{E}$. For each stage at each step, the initial proposal is round $j = 0$ and the runtime permits at most B reflection revisions, indexed by $j = 1, \ldots, B$. Every rejected proposal is blocked before deployment, revised, and re-evaluated; if none of the proposals at rounds $j = 0, \ldots, B$ is accepted, the runtime aborts the current execution. Under a fixed λ, the verifiers, gate, and reflection operators induce $\xi(\lambda)$. Let $L(\lambda) \in [0,1]$ be the full runtime system loss, let $R(\lambda)$ be the corresponding population system risk, and let $U(\lambda)$ be the expected task utility of the same runtime, the quantity a deployment trades against risk. Appendix A collects the notation of this section and of Section 3.3. We write $\omega \in \Omega$ for the exogenous randomness of one execution, namely the decoding seeds of the policy, verifier and reflection operator together with environment stochasticity, drawn once per instance and reused across configurations, so that $\xi(\lambda) = \Xi(x, \omega; \lambda)$ is a deterministic function of (x, ω) on all of Λ.

2.2 Conformal Risk Control (CRC)

Conformal Risk Control (CRC) (Angelopoulos et al., 2022) provides a distribution free calibration principle for certifying the expected system loss of a deployed Agent Runtime. Given an independent and identically distributed calibration set and a test instance from the same distribution, CRC considers a family of outputs indexed by a conservativeness parameter $\lambda \in \Lambda$, where larger λ induces more conservative behavior and a non increasing loss. CRC applies to any bounded monotone loss. Based on the calibration losses, it selects a data dependent configuration $\widehat{\lambda}$ and guarantees that the expected loss on a fresh test instance is controlled at the target risk level α:

$$\mathbb{E}\Big[L_{n+1}(\widehat{\lambda})\Big] \leq \alpha.$$

In our setting, λ corresponds to the Agent Runtime configuration of CertAgent, and $L_{n+1}(\widehat{\lambda})$ corresponds to the system level loss over a fresh deployment trajectory. Therefore, CRC provides the statistical foundation for certifying Agent Runtime Fragility at the entire system level. Concretely, the guarantee requires the realized losses to satisfy three conditions: **(C1)** they are exchangeable across instances; **(C2)** for every instance, $L_i(\cdot)$ is bounded and non increasing in λ; and **(C3)** the most conservative candidate already meets the target, $L_i(\lambda_{\max}) \leq \alpha$ almost surely, which is the condition under which the selection set is non empty. Right continuity in λ is required in general and is vacuous on a finite grid. Section 3.3 establishes all three conditions for the Agent Runtime family rather than assuming them.

2.3 Problem Formulation

ReAct style agents (Yao et al., 2022) often fail through sequential and compounding runtime errors (Zhu et al., 2025; Pan et al., 2025; Barke et al., 2026; Zhang et al., 2025). An early reasoning or action error may affect the downstream trajectory and eventually lead to task failure. In the reasoning stage, such errors mainly arise from inappropriate runtime control or insufficient reasoning support. In the action stage, they mainly arise from argument errors or execution errors. We refer to this phenomenon as *Agent Runtime Fragility*. A certificate is informative only if the certified quantity is one that runtime control can influence. Raw task failure is not such a quantity, since an agent may fail because the instance exceeds the capability of the policy, which runtime gating cannot correct. We therefore certify the runtime attributable component of failure: the exposure the runtime accumulates by deploying proposals its own verifier judges unreliable, together with the executions it abandons.

Definition 1 (Agent Runtime Fragility Certification)**.** *Given a controlled Agent Runtime trajectory $\xi(\lambda)$ induced by a ReAct agent under runtime configuration $\lambda \in \Lambda$, we measure Agent Runtime Fragility by a system level loss $L(\lambda) = \ell(\xi(\lambda)) \in [0,1]$. Let $\mathcal{D}(\xi) \subseteq \{(t,m) : 1 \leq t \leq T,\ m \in \{R, A\}\}$ be the set of deployed step–stage proposals along the trajectory, and let q_t^m be the verifier*

score of the proposal deployed at step t and stage m. An execution that completes incurs the average verifier exposure of its deployed proposals,

$$\ell(\xi) = \frac{1}{|\mathcal{D}(\xi)|} \sum_{(t,m)\in\mathcal{D}(\xi)} (1 - q_t^m),$$

while an execution that exhausts its reflection budget and aborts is assigned the maximal loss $\ell(\xi) = 1$. The corresponding system risk is $R(\lambda) = \mathbb{E}[L(\lambda)]$. Given a target risk level $\alpha \in (0, 1)$, our goal is to certify an upper bound on $R(\lambda)$ for a fixed configuration, or equivalently the set of configurations whose certified bound is at most α.

The average in Definition 1 normalizes verifier exposure across trajectories with different numbers of deployed proposals. The second convention is not a normalization choice: a loss that scored only completed executions would record the removal of an execution as an improvement and would exhibit a spurious monotone decrease in λ.

Verifier-proxy Assumption. Assume that a calibrated verifier provides an informative measure of proposal reliability. *Agent Runtime Fragility certification* then controls the verifier-induced trajectory loss in Definition 1, rather than raw task success. Its validity is conditional on verifier quality and calibration, evaluated in Appendix C. Our goal is therefore to control Agent Runtime Fragility over the *entire* ReAct runtime rather than at an isolated intermediate step. To operationalize it, we decompose the runtime into reasoning and action stages, which capture different failure modes and enable stage specific verification and reflection. The boundary is a modeling device rather than a claim about the runtime, since planning and execution are coupled throughout the ReAct loop; it is introduced to provide a well defined interface for modeling, calibration, and control, while the certification target remains the system level loss L over the *entire* deployment trajectory.

3 METHODOLOGY

3.1 OVERVIEW: CERTAGENT FRAMEWORK

Our goal is to control *Agent Runtime Fragility* at the system level. Since the reasoning and action stages exhibit different failure modes, CertAgent uses a stage structured runtime control mechanism while keeping the certification target on the *entire* Agent Runtime. Calibration runs over the configuration grid of Section 2.1: for a fixed λ we certify that the Agent Runtime risk does not exceed α, and for a target α we return the configurations that are valid for deployment.

CertAgent controls runtime conservativeness through a Verifier Subagent and a Gate Module. At step t and stage m, the Verifier Subagent returns $(q_t^m, e_t^m) = V_m(s_t, z_t^m)$, where e_t^m explains the main reason for low confidence. An accepted proposal is deployed; a rejected proposal is blocked before deployment, revised by $\mathrm{Reflect}_m(s_t, z_t^m, e_t^m)$, and re-evaluated by the same verifier, which generates the reflection sequence formalized in Lemma 1. For each stage at each step, the initial proposal is round $j = 0$ and at most B reflection revisions are allowed. This allow or reflect loop continues until a proposal is accepted or all rounds $j = 0, \ldots, B$ have been rejected, in which case the execution terminates and Definition 1 charges the maximal loss. The two stages share this structure but instantiate different proposal spaces, failure modes, verifier scores, and reflection feedback.

3.2 CERTAGENT INSTANTIATION FOR REACT RUNTIME STAGES

CertAgent uses a shared verifier, gate, and reflection structure for both stages, instantiated with stage specific proposal spaces and failure modes. Since agent benchmarks provide only final task outcomes rather than ground truth intermediate decisions, CertAgent does not certify reasoning or action labels; it verifies whether each stage proposal is reliable under the current runtime state.

Reasoning Stage. The reasoning proposal is $z_t^R = r_t$ and the stage targets $\mathcal{F}_R = \{\text{Control Error}, \text{ Reasoning Insufficiency}\}$. A *Control Error* occurs when the proposal induces an inappropriate runtime transition under the current state; a *Reasoning Insufficiency* occurs when it is not sufficiently supported, for example due to missing constraints, inconsistent reasoning, insufficient evidence, or unresolved ambiguity. The verifier returns $(q_t^R, e_t^R) = V_R(s_t, r_t)$ with

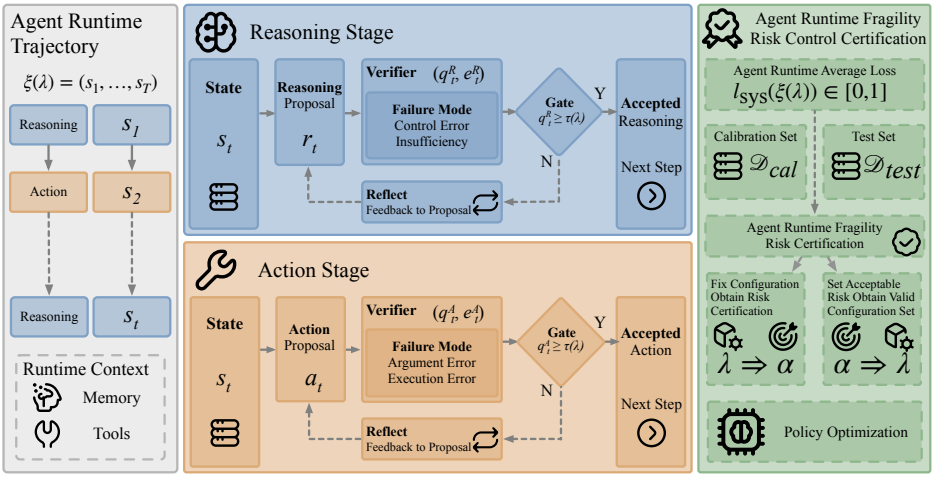


Figure 1: Overview of *CertAgent*. The left panel shows the runtime trajectory with context, memory, and tools. The middle shows control: the Verifier Subagent scores proposals, the Gate Module controls conservativeness through λ, and reflection revises rejected proposals. The right shows trajectory level calibration for certifying a fixed configuration or a configuration set at a target risk, with policy optimization improving certification.

$q_t^R = \Gamma_R\big(v_R^{\mathrm{ctrl}}(s_t, r_t),\, v_R^{\mathrm{insuf}}(s_t, r_t)\big)$, where $v_R^{\mathrm{ctrl}}, v_R^{\mathrm{insuf}}$ measure reliability against the two modes and $\Gamma_R : [0,1]^2 \to [0,1]$ aggregates them.

Action Stage. The action proposal is $z_t^A = a_t$ and the stage targets $\mathcal{F}_A = \{\text{Argument Error}, \text{Execution Error}\}$. An *Argument Error* occurs when the action arguments are missing, incomplete, malformed, or incorrect; an *Execution Error* occurs when the action may fail during execution or is unlikely to realize the intended effect. The verifier returns $(q_t^A, e_t^A) = V_A(s_t, a_t)$ with $q_t^A = \Gamma_A\big(v_A^{\mathrm{arg}}(s_t, a_t),\, v_A^{\mathrm{exec}}(s_t, a_t)\big)$, defined analogously.

3.3 Certified Agent Runtime Fragility Risk Control

The two stages jointly induce controlled Agent Runtimes indexed by λ, while certification targets their system loss. For calibration instance i, let $L_i(\lambda) = \ell(\xi_i(\lambda)) \in [0,1]$ and $\widehat{R}_n(\lambda) = n^{-1}\sum_{i \leq n} L_i(\lambda)$. Conditions (C2) and (C3) of Section 2.2 are not automatic. Stricter gating may lower verifier exposure through higher accepted scores, but may also increase reflection and budget exhaustion, which incurs maximal loss. Hence $L_i(\lambda)$ need not be non increasing. If $\Pr[\xi_i(\lambda_K) \text{ aborts}] > 0$, then $L_i(\lambda_K) = 1$ with positive probability, violating the almost sure requirement in (C3) for every $\alpha < 1$. We obtain (C1) from the sampling design and coupling of Section 2.1, enforce (C2) with a non increasing majorant, and satisfy (C3) by extending the grid to $\Lambda^+ = \Lambda \cup \{\lambda_\dagger\}$ with an abstention option $\lambda_\dagger \succ \lambda_K$. All subsequent statements apply to Λ^+.

Exchangeability (C1). Because calibration uses the entire path $\lambda \mapsto L_i(\lambda)$, (C1) requires these random functions to be exchangeable. The pairs (x_i, ω_i) are exchangeable, and the coupling reuses ω_i across Λ^+, making each path the image of (x_i, ω_i) under one fixed map. Applying this map coordinatewise preserves exchangeability. Independence from the calibration data follows from the training–calibration split of Section 3.4. Appendix B verifies both properties.

Monotonicity (C2). Under a configuration independent reflection sequence, the deployed score is monotone within a step, but the trajectory loss need not be. The following lemma formalizes the former; a majorant enforces the latter.

Lemma 1 (Pathwise monotonicity of the deployed score)**.** *Fix an instance (x, ω), a stage m, and a state s. Starting from $z^{(0)}$, define the reflection sequence*

$$(q^{(j)}, e^{(j)}) = V_m\big(s, z^{(j)}\big), \qquad z^{(j+1)} = \mathrm{Reflect}_m\big(s, z^{(j)}, e^{(j)}\big),$$

which depends on $(s, z^{(j)}, e^{(j)})$ but not on λ. Let $j(\lambda) = \min\big\{j \leq B :\ q^{(j)} \geq \tau(\lambda)\big\}$ be the first accepted round, with $j(\lambda) = \infty$ if none is accepted, in which case the step aborts and incurs maximal

loss. For any $\lambda \le \lambda'$ in Λ with $j(\lambda') < \infty$,

$$j(\lambda) \le j(\lambda') \qquad \textit{and} \qquad q^{(j(\lambda'))} \ \ge \ q^{(j(\lambda))}.$$

Within a common reflection sequence, a higher threshold can only delay acceptance and cannot lower the deployed score, so the per step exposure $1 - q_t$ is non increasing when acceptance occurs within budget. This does not imply trajectory level monotonicity because different proposals induce different subsequent states. Moreover, our reflection prompt includes the acceptance threshold, so the shared sequence hypothesis need not hold exactly. The lemma explains the observed trend but is not assumed; (C2) is enforced by construction.

For each instance, define the least non increasing majorant of L_i by

$$\widetilde{L}_i(\lambda) \ = \ \max_{\lambda' \in \Lambda^+,\ \lambda' \ge \lambda} L_i(\lambda'), \qquad \lambda \in \Lambda^+, \qquad \widetilde{R}_n(\lambda) = \frac{1}{n}\sum_{i=1}^{n} \widetilde{L}_i(\lambda).$$

By construction, $\widetilde{L}_i \in [0,1]$ is non increasing, dominates L_i pointwise, and remains exchangeable as a fixed measurable function of (x_i, ω_i). Certifying $\widetilde{L}$ therefore certifies L and enforces (C2). Replacing $\widehat{R}_n$ by $\widetilde{R}_n$ increases the empirical risk by the reported gap $\widetilde{R}_n - \widehat{R}_n$, and the CRC bound by $\frac{n}{n+1}(\widetilde{R}_n - \widehat{R}_n)$. An abort at a larger configuration sets the majorant to 1 at every smaller configuration, so we use a grid with low abort rates and report them for all candidates.

Abstention (C3). To represent an unattainable target, we equip the abstention option with

$$\ell\big(\xi(\lambda_\dagger)\big) \equiv 0, \qquad U(\lambda_\dagger) = 0,$$

so no execution is attempted. Zero loss records zero deployment exposure, while $U(\lambda_\dagger) = 0$ charges the cost on the utility axis. Because $\lambda_\dagger$ is the largest element of Λ^+, $\widetilde{L}_i(\lambda_\dagger) = L_i(\lambda_\dagger) = 0 \le \alpha$ almost surely, establishing (C3). The certificate is informative through whether it returns a deployable configuration and the utility attained; otherwise CertAgent abstains, analogous to a reject option (Chow, 1970; Geifman & El-Yaniv, 2017). Appendix B.1 gives the full accounting.

Theorem 1 (Certified Agent Runtime Fragility Risk)**.** *Let $(x_i, \omega_i)_{i=1}^{n+1}$ be exchangeable and let $\alpha \in [\frac{1}{n+1}, 1)$. Define the certified configuration set and the CRC selection*

$$\widehat{\mathcal{F}}(\alpha) = \Big\{\lambda \in \Lambda^+ : \ \tfrac{n}{n+1}\widetilde{R}_n(\lambda) + \tfrac{1}{n+1} \le \alpha\Big\}, \qquad \widehat{\lambda}(\alpha) = \min \widehat{\mathcal{F}}(\alpha).$$

Then $\lambda_\dagger \in \widehat{\mathcal{F}}(\alpha)$, so $\widehat{\mathcal{F}}(\alpha)$ is non empty and $\widehat{\mathcal{F}}(\alpha) = \{\lambda \in \Lambda^+ : \lambda \ge \widehat{\lambda}(\alpha)\}$, and

$$\mathbb{E}\big[L_{n+1}(\lambda)\big] \ \le \ \alpha \qquad \textit{for every } \lambda \in \widehat{\mathcal{F}}(\alpha).$$

Theorem 1 supports two deployment views. For a fixed λ, its certified level is $\frac{n}{n+1}\widetilde{R}_n(\lambda) + \frac{1}{n+1}$; for a fixed target α, any element of $\widehat{\mathcal{F}}(\alpha)$ may be selected, for example by validation utility. If $\widehat{\mathcal{F}}(\alpha) = \{\lambda_\dagger\}$, no deployable candidate is certified and CertAgent abstains.

The preceding guarantee is in expectation, whereas deployment uses one realized calibration set. Exchangeability also yields an exact rank based guarantee because monotonicity represents each loss path by the smallest configuration meeting the target.

Theorem 2 (Exact certified configuration set)**.** *Let $(x_i, \omega_i)_{i=1}^{n+1}$ be exchangeable, let $\alpha \in (0,1)$ and $\delta \in [\frac{1}{n+1}, 1)$, and set $k = \lceil (1-\delta)(n+1) \rceil$. For each instance let $\widehat{\lambda}_i(\alpha) = \min\{\lambda \in \Lambda^+ : \widetilde{L}_i(\lambda) \le \alpha\}$, and let $\lambda^\star_\delta(\alpha)$ be the k-th smallest of $\widehat{\lambda}_1(\alpha), \dots, \widehat{\lambda}_n(\alpha)$. Then*

$$\Pr\Big(L_{n+1}(\lambda) \le \alpha \ \textit{for every } \lambda \in \Lambda^+ \textit{ with } \lambda \ge \lambda^\star_\delta(\alpha) \Big) \ \ge \ 1 - \delta.$$

Corollary 1 (Duality of the two deployment views)**.** *Let $\widehat{\alpha}_\delta(\lambda)$ be the k-th smallest of $\widetilde{L}_1(\lambda), \dots, \widetilde{L}_n(\lambda)$. Then $\widehat{\alpha}_\delta$ is non increasing on Λ^+ and, for every $\alpha \in (0,1)$,*

$$\big\{\lambda \in \Lambda^+ : \ \widehat{\alpha}_\delta(\lambda) \le \alpha\big\} \ = \ \big\{\lambda \in \Lambda^+ : \ \lambda \ge \lambda^\star_\delta(\alpha)\big\}.$$

Thus, the fixed configuration and target risk views are dual readings of the same order statistic and share the event in Theorem 2.

Theorem 2 controls the next deployment loss with probability $1-\delta$, whereas Theorem 1 controls expected loss. Its order statistic adapts to the calibration losses, incurs no explicit multiplicity penalty in K, and applies simultaneously to the entire certified set by Corollary 1. The error level must satisfy $\delta \geq 1/(n+1)$. To audit the calibration construction, we report candidatewise abort rates, the fraction of already monotone loss paths, and the majorant gap $\widetilde{R}_n - \widehat{R}_n$, as defined in Appendix C.

3.4 Policy Optimization for Effective System Certification

Calibration certifies a fixed family of controlled runtimes; it does not improve the underlying policy. We therefore optimize the policy before calibration, so that the family being certified already has lower runtime risk. Under a fixed λ, the policy π_θ together with the verifiers, gate, and reflection mechanisms induces a trajectory $\xi^{\pi_\theta}(\lambda) \sim P^{\lambda}_{\pi_\theta}$ and the system loss $L_{\pi_\theta}(\lambda) = \ell(\xi^{\pi_\theta}(\lambda)) \in [0,1]$, which is exactly the certified object. We optimize the risk penalized objective

$$J_\beta(\pi_\theta;\lambda) = U_{\pi_\theta}(\lambda) - \beta\, R_{\pi_\theta}(\lambda), \qquad \beta > 0,$$

where $U_{\pi_\theta}(\lambda)$ is the expected task utility and $R_{\pi_\theta}(\lambda)$ the expected system level runtime risk, and β controls the tradeoff between them. Optimization uses the training split only and the certificate is computed on an independent calibration split, which is what preserves exchangeability for the optimized runtime. Proposition 1 in Appendix B states the consequence for certification. If the optimized runtime loss is stochastically dominated by the original one as a random function on Λ^+, then at every configuration the certified level of Theorem 2 decreases stochastically, at every target level the certified set expands stochastically, and the CRC selection of Theorem 1 becomes no more conservative. These are statements about the certificate a user obtains, in contrast to a comparison of two empirical risks, which would not by itself establish any property of the certificate.

4 Experiment

In this section, we focus on three core research questions (RQ). **RQ1**: How effective is CertAgent in improving overall agent performance? **RQ2**: How effective is the proposed Runtime Risk Control mechanism in providing practical certification guarantees? **RQ3**: How does policy optimization affect certification performance, and what runtime efficiency does CertAgent achieve?

4.1 Experimental Setup

Base Models. We consider both open and closed source backbones. The open source models are Llama-3.1-8B (Grattafiori et al., 2024) and Gemma-3-4B (Team et al., 2024), which provide reproducible backbones with accessible weights; the closed source model is GPT-5.5 (Singh et al., 2025). We apply LoRA (Hu et al., 2021) fine tuning to the open source models and use the closed source model for inference based evaluation.

Agent Baselines. To evaluate CertAgent in realistic agentic settings, we compare against representative agent systems, including smolagents (Roucher et al., 2025), AutoAgent (Tang et al., 2025), and our proposed CertAgent. These baselines allow us to compare standard agent orchestration pipelines with our certified runtime control framework under the same benchmark environment. For benchmark compatibility, all agent systems are equipped with the same Live-SWE harness (Xia et al., 2025), ensuring that they share a consistent task solving interface and operate toward the same benchmark objectives.

Benchmark and Metrics. We conduct experiments on SWE-Bench Lite (Jimenez et al., 2023), which evaluates an agent's ability to resolve real-world GitHub issues through code understanding, repository navigation, tool use, and patch generation. We split the benchmark into a training set for LoRA fine-tuning, a calibration set of $n = 60$ instances for certification, and a test set for final evaluation. Downstream task performance is measured by success rate, defined as the percentage of instances for which the generated patch resolves the issue under the benchmark evaluation harness. For certification, we report the certified Agent Runtime Fragility risk bound α and task utility, where utility is the fraction of successfully solved instances. Runtime efficiency is measured by average token usage and inference time.

4.2 Overall Effectiveness of CertAgent (RQ1)

Table 1: Downstream task performance for **RQ1** on the 60-instance test split (success rate, %). Best results are in **bold**, and second-best results are underlined.

Method	Llama-3.1-8B	Gemma-3-4B	GPT-5.5
Smolagent	18.3	11.7	85.0
AutoAgent	21.7	13.3	88.3
CertAgent	23.3	15.0	**91.7**
CertAgent + RL	**30.0**	**23.3**	N/A

We evaluate CertAgent on SWE-Bench Lite (Jimenez et al., 2023) against representative baseline agents, instantiate it with different LLM backbones to examine generalization across models, and further perform policy optimization for the open-source models under the same architecture. As shown in Table 1, CertAgent achieves strong downstream performance despite the limited capability of small open-source models, and reaches its best results with the closed-source GPT-5.5 backbone at reasoning effort xHigh. We attribute this to the framework design: the Verifier Subagent assesses each runtime step, and combined with the Gate and Reflection mechanisms it reduces hallucination in the reasoning stage and tool-use errors in the action stage, at an additional computational cost analyzed below. Fine-tuning CertAgent with LoRA (Hu et al., 2021) yields further gains, consistent with the policy learning to improve the quality of both reasoning and action decisions during the runtime.

4.3 Certification Performance of Runtime Risk Control (RQ2)

Table 2: *Agent Runtime Fragility* certification for **RQ2** under Theorem 1, with $n = 60$ and $K = 5$. Panel (a) reports certified risk and utility by configuration; panel (b) reports certified sets, deployments, and utility by target risk. A dash denotes certified abstention.

(a) Fix configuration λ, report risk α

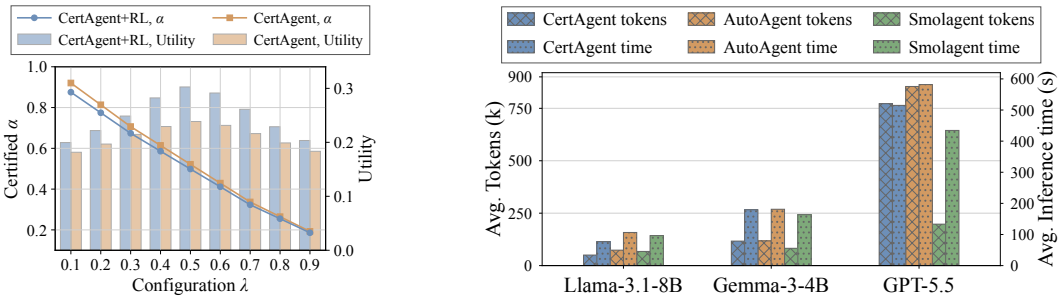
Target λ	Certified α	Utility
0.10	0.85	0.18
0.30	0.67	0.22
0.50	0.48	0.23
0.70	0.29	0.22
0.90	0.13	0.18

(b) Fix risk α, report configuration λ

Target α	Certified $\widehat{\mathcal{F}}(\alpha)$	Deployed λ^*	Utility
0.10	—	abstain	—
0.30	{0.70, 0.90}	0.70	0.22
0.50	{0.50, 0.70, 0.90}	0.50	0.23
0.70	{0.30, 0.50, 0.70, 0.90}	0.50	0.23
0.90	{0.10, 0.30, 0.50, 0.70, 0.90}	0.50	0.23

This section studies the practical certification performance of CertAgent, a key advantage over existing certified methods. Table 2 reports both deployment views of Theorem 1 on the same calibration set. These are two readings of one guarantee: a user with a default configuration reads its certified risk level α, and a user with an acceptable α reads the set of configurations meeting it. When the configuration is fixed (Table 2a), a stricter λ receives a smaller certified risk bound, and the bound decreases monotonically over the tested grid. Lemma 1 accounts for this: raising the acceptance threshold makes the runtime keep reflecting until it reaches a proposal the verifier scores at least as high, which lowers the exposure accumulated along the deployed trajectory. The trend should not be extrapolated beyond the tested grid. Once $\tau(\lambda)$ approaches the upper end of the verifier score range, executions begin to exhaust the reflection budget and abort, and Definition 1 charges those executions the maximal loss, so the certified bound turns upward. We therefore place the grid inside the regime where the abort rate is negligible, and report the diagnostics of Section 3.3 in Appendix C. In the reverse view (Table 2b) the user fixes α and reads off $\widehat{\mathcal{F}}(\alpha)$, which grows to include more aggressive configurations as the requirement loosens. Two consequences of Theorem 1 are visible. First, the guarantee holds for every element of the certified set and not only for the selection $\widehat{\lambda}(\alpha)$, so the deployed configuration may be chosen inside the set by utility; at $\alpha = 0.70$ and $\alpha = 0.90$ this selects $\lambda = 0.50$ rather than the most aggressive certified candidate. Second, at $\alpha = 0.10$ the certified set contains no deployable configuration, since even the most conservative candidate carries a certified bound of 0.13, and CertAgent abstains. Returning $\lambda = 0.90$ at this level would assert a guarantee that its own certificate contradicts. Appendix Table 4 reports the corresponding 55th-order high-probability certificate at $\delta = 0.1$ together with the calibration diagnostics.

Turning to utility, the values in Table 2 are quasi concave over the grid and peak at $\lambda = 0.50$. The two ends degrade for different reasons: an aggressive configuration deploys proposals the verifier



(a) Optimization Effect

(b) Runtime Efficiency

Figure 2: **Left**: comparison of CertAgent before and after policy optimization in terms of certified risk α and utility across runtime configurations. **Right**: efficiency comparison between CertAgent and agent baselines, measured by average token usage and average inference time.

judges unreliable, whereas a conservative one spends the reflection budget on repeated revisions. Certified risk and utility therefore do not move together over the grid. The free selection permitted by Theorem 1 is therefore operationally relevant: the utility maximizing element of $\widehat{\mathcal{F}}(\alpha)$ is in general an interior point of the certified set rather than its most aggressive or most conservative element. Overall, CertAgent issues an actionable certificate: a certified risk level for a fixed configuration, a certified set with free selection inside it for a target level, and an explicit abstention when the target is not attainable with the available calibration data.

4.4 Policy Optimization Effects and Runtime Efficiency (RQ3)

We analyze CertAgent from two perspectives: the effect of policy optimization on certification and utility, and the runtime efficiency of CertAgent compared with agent baselines. We first examine the effect of policy optimization. As shown in Figure 2a, we evaluate nine candidate runtime configurations and report the corresponding certified risk bound α. On this calibration sample, the aggregate certified bound is lower at every evaluated configuration. Consequently, the empirical certified set obtained from this sample expands rather than merely shifting its selected operating point; this aggregate comparison does not by itself verify the pathwise domination hypothesis of Proposition 1. The improvement is larger under smaller, more aggressive configurations, indicating that policy optimization is most effective at tightening the guarantee where the runtime relies least on gating. We also observe that utility improves after policy optimization. The improvement is limited under the most aggressive and most conservative configurations, while more moderate configurations obtain clear utility gains. This suggests that policy optimization is most effective when the runtime configuration provides a balanced space between task utility and risk control. As shown in Figure 2b, the average token consumption and inference time of CertAgent are close to those of the bare agent skeleton SMOLAGENTS and consistently better than the more complex AUTOAGENT. Policy optimization therefore improves both certification effectiveness and task utility, while runtime efficiency remains comparable to agent baselines.

5 Conclusion

In this paper, we proposed *CertAgent*, a framework for Agent Runtime Fragility Risk Control. We formulated Agent Runtime Fragility as system level risk over the entire Agent Runtime, charging the maximal loss to abandoned executions. CertAgent combines a Verifier Subagent, a Gate Module, and reflection to detect and revise unreliable runtime proposals, and establishes rather than assumes the conditions calibration requires, via a non increasing majorant and an abstention option, yielding an expectation certificate valid across the certified set and an exact finite sample certificate unifying both views. Policy optimization further improves the certificate itself, not only the empirical risk. Experiments on representative agent benchmarks demonstrate that CertAgent improves runtime reliability, supports practical risk utility tradeoffs, and maintains strong runtime efficiency, making it a practical step toward certified deployment of LLM agents.

AI USE STATEMENT

AI tools were used solely to polish the language and improve the presentation of this manuscript. They were not used for any other purpose, including developing the research ideas or methodology, conducting the experiments or analysis, or generating or modifying the reported results and conclusions. The authors reviewed and verified the entire manuscript, including all AI-assisted edits, and take full responsibility for its content.

ETHICS STATEMENT

This work evaluates software engineering agents on the publicly available SWE-Bench Lite benchmark and involves no human participants, interventions, or collection of new personal data. The experiments use publicly available software engineering tasks and are conducted in a controlled Live-SWE evaluation environment to reduce unintended interactions with external systems. Nevertheless, runtime certification does not eliminate all potential failures or harms. Any real-world deployment, especially in safety-critical settings, should therefore include task-specific validation, human oversight, appropriate access controls, and compliance with applicable licenses and policies.

REFERENCES

Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B Divya. Agentic ai: Autonomous intelligence for complex goals—a comprehensive survey. *IEEe Access*, 13:18912–18936, 2025.

Anastasios N Angelopoulos, Stephen Bates, Adam Fisch, Lihua Lei, and Tal Schuster. Conformal risk control. *arXiv preprint arXiv:2208.02814*, 2022.

Robert L Axtell and J Doyne Farmer. Agent-based modeling in economics and finance: Past, present, and future. *Journal of Economic Literature*, 63(1):197–287, 2025.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. Agentrx: Diagnosing ai agent failures from execution trajectories. *arXiv preprint arXiv:2602.02475*, 2026.

Varun Pratap Bhardwaj. Agent behavioral contracts: Formal specification and runtime enforcement for reliable autonomous ai agents. *arXiv preprint arXiv:2602.22302*, 2026.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. *arXiv preprint arXiv:2410.13232*, 2024.

C Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. Agent-rlvr: Training software engineering agents via guidance and environment rewards. *arXiv preprint arXiv:2506.11425*, 2025.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, 2017.

Alireza Ghafarollahi and Markus J Buehler. Sciagents: automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523, 2025.

Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. Plotgen: multi-agent llm-based scientific data visualization via multimodal retrieval feedback. In *Companion Proceedings of the ACM on Web Conference 2025*, pp. 1672–1676, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Mourad Gridach, Jay Nanavati, Khaldoun Zine El Abidine, Lenon Mendes, and Christina Mack. Agentic ai for scientific discovery: A survey of progress, challenges, and future directions. *arXiv preprint arXiv:2503.08979*, 2025.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, 2024.

Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 2025.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, et al. Data interpreter: An llm agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19796–19821, 2025.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14281–14290, 2024.

Soodeh Hosseini and Hossein Seilani. The role of agentic ai in shaping a smart future: A systematic review. *Array*, 26:100399, 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

Chengyou Jia, Minnan Luo, Zhuohang Dang, Qiushi Sun, Fangzhi Xu, Junlin Hu, Tianbao Xie, and Zhiyong Wu. Agentstore: Scalable integration of heterogeneous agents as specialized generalist computer assistant. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 8908–8934, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025.

Nalan Karunanayake. Next-generation agentic ai for transforming healthcare. *Informatics and Health*, 2(2):73–83, 2025.

Roham Koohestani. Agentguard: Runtime verification of ai agents. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pp. 74–77. IEEE, 2025.

Jakub Lála, Odhran O'Donoghue, Aleksandar Shtedritski, Sam Cox, Samuel G Rodriques, and Andrew D White. Paperqa: Retrieval-augmented generative agent for scientific research. *arXiv preprint arXiv:2312.07559*, 2023.

Christoph R Landolt, Christoph Würsch, Roland Meier, Alain Mermoud, and Julian Jang-Jaccard. Multi-agent reinforcement learning in cybersecurity: From fundamentals to applications. *arXiv preprint arXiv:2505.19837*, 2025.

Haohang Li, Yupeng Cao, Yangyang Yu, Shashidhar Reddy Javaji, Zhiyang Deng, Yueru He, Yuechen Jiang, Zining Zhu, Kp Subbalakshmi, Jimin Huang, et al. Investorbench: A benchmark for financial decision-making tasks with llm-based agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2509–2525, 2025.

Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey. *ACM Transactions on Software Engineering and Methodology*, 2024.

Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. Evaluation and benchmarking of llm agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6129–6139, 2025.

Michael Moritz, Eric Topol, and Pranav Rajpurkar. Coordinated ai agents for advancing healthcare. *Nature Biomedical Engineering*, 9(4):432–438, 2025.

San Murugesan. The rise of agentic ai: implications, concerns, and the path forward. *IEEE Intelligent Systems*, 40(2):8–14, 2025.

Liangbo Ning, Ziran Liang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Wenqi Fan, Xiao-yong Wei, Shanru Lin, Hui Liu, Philip S Yu, et al. A survey of webagents: Towards next-generation ai agents for web automation with large foundation models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6140–6150, 2025.

Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, et al. Why do multiagent systems fail? In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025.

Shuo Ren, Can Xie, Pu Jian, Zhenjiang Ren, Chunlin Leng, and Jiajun Zhang. Towards scientific intelligence: A survey of llm-based scientific agents. *arXiv preprint arXiv:2503.24047*, 2025.

Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 'smolagents': a smol library to build great agentic systems. https://github.com/huggingface/smolagents, 2025.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. *arXiv preprint arXiv:2309.15817*, 2023.

Bikash Saha and Sandeep Kumar Shukla. Malgen: A generative agent framework for modeling malicious software in cybersecurity. *arXiv preprint arXiv:2506.07586*, 2025.

Johannes Schneider. Generative to agentic ai: Survey, conceptualization, and challenges. *arXiv preprint arXiv:2504.18875*, 2025.

Yonadav Shavit, Sandhini Agarwal, Miles Brundage, Steven Adler, Cullen O'Keefe, Rosie Campbell, Teddy Lee, Pamela Mishkin, Tyna Eloundou, Alan Hickey, et al. Practices for governing agentic ai systems. *Research Paper, OpenAI*, 2023.

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.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024.

Jiabin Tang, Tianyu Fan, and Chao Huang. Autoagent: A fully-automated and zero-code framework for llm agents. *arXiv preprint arXiv:2502.05957*, 2025.

Xunzhu Tang, Kisub Kim, Yewei Song, Cedric Lothritz, Bei Li, Saad Ezzini, Haoye Tian, Jacques Klein, and Tegawendé F Bissyandé. Codeagent: Autonomous communicative agents for code review. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11313, 2024.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Hongru Wang, Cheng Qian, Manling Li, Jiahao Qiu, Boyang Xue, Mengdi Wang, Heng Ji, and Kam-Fai Wong. Toward a theory of agents as tool-use decision-makers. *arXiv preprint arXiv:2506.00886*, 2025a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Wenxuan Wang, Zizhan Ma, Zheng Wang, Chenghan Wu, Jiaming Ji, Wenting Chen, Xiang Li, and Yixuan Yuan. A survey of llm-based agents in medicine: How far are we from baymax? *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10345–10359, 2025b.

Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. Cybergym: Evaluating ai agents' cybersecurity capabilities with real-world vulnerabilities at scale. *arXiv e-prints*, pp. arXiv–2506, 2025c.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-swe-agent: Can software engineering agents self-evolve on the fly? *arXiv preprint arXiv:2511.13646*, 2025.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

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.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *European Conference on Computer Vision*, pp. 240–255. Springer, 2024.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13643–13658, 2024.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025.

A NOTATION

Table 3 collects the symbols of Sections 2.1 and 3.3. A hat marks a quantity computed from the calibration sample, a tilde marks the non increasing majorant of Section 3.3, and a star marks an order statistic across calibration instances. Throughout, ℓ, L and R denote the system level loss and risk of Definition 1 over the entire runtime trajectory; no per step loss is defined, so they carry no further decoration.

Table 3: Notation used in the runtime model and in the certification.

Runtime	
$x,\ \omega$	task instance; exogenous randomness of one execution, drawn once per instance and reused across Λ^+
$\xi(\lambda)$	controlled runtime trajectory $\Xi(x, \omega; \lambda)$, a deterministic function of (x, ω) on Λ^+
$t,\ T,\ s_t$	step index, horizon, runtime state
$m \in \{R, A\}$	runtime stage, reasoning or action
z_t^m	stage proposal, $z_t^R = r_t$ and $z_t^A = a_t$
$V_m,\ q_t^m,\ e_t^m$	stage verifier; verifier score in $[0, 1]$; textual feedback
$\mathrm{Reflect}_m,\ B,\ j$	reflection operator; maximum reflection revisions per stage and step; proposal round index, with $j = 0$ denoting the initial proposal
$\mathcal{D}(\xi)$	set of deployed step–stage pairs (t, m) along ξ
Configuration	
$\lambda,\ \Lambda$	runtime configuration; the candidate grid $\{\lambda_1 < \cdots < \lambda_K\}$ fixed before calibration
$\tau(\lambda),\ G(q; \lambda)$	non decreasing threshold map; gate $\mathbf{1}\{q \geq \tau(\lambda)\}$
$\lambda_\dagger,\ \Lambda^+$	abstention option, $\lambda_\dagger \succ \lambda_K$; enlarged candidate set $\Lambda \cup \{\lambda_\dagger\}$
Loss, risk, and utility	
$\ell(\xi)$	system loss of a trajectory (Definition 1)
$L_i(\lambda)$	realized system loss $\ell(\xi_i(\lambda))$ of calibration instance i
$\widetilde{L}_i(\lambda)$	its least non increasing majorant $\max_{\lambda' \geq \lambda} L_i(\lambda')$ on Λ^+
$\widehat{R}_n,\ \widetilde{R}_n$	empirical means of L_i and of $\widetilde{L}_i$ over the n calibration instances
$R(\lambda),\ U(\lambda)$	population system risk; expected task utility
Certification	
$n,\ \alpha,\ \delta,\ k$	calibration size; target risk level; error level; rank $\lceil (1-\delta)(n+1) \rceil$
$\widehat{\mathcal{F}}(\alpha),\ \widehat{\lambda}(\alpha)$	certified configuration set at level α; its least element, the CRC selection
$\widehat{\lambda}_i(\alpha)$	smallest configuration at which instance i already satisfies $\widetilde{L}_i \leq \alpha$
$\lambda^\star_\delta(\alpha)$	k-th smallest of $\widehat{\lambda}_1(\alpha), \ldots, \widehat{\lambda}_n(\alpha)$ (Theorem 2)
$\widehat{\alpha}_\delta(\lambda)$	k-th smallest of $\widetilde{L}_1(\lambda), \ldots, \widetilde{L}_n(\lambda)$, the certified level at λ
Policy optimization	
$\pi_\theta,\ \beta,\ J_\beta$	policy and its parameters; risk penalty weight; penalized objective

B THEORETICAL PROOFS

Throughout this appendix we work under the runtime model of Section 2.1. Because Λ^+ is finite and totally ordered, every function of λ below is a step function and no continuity condition is required. We fix any order preserving injection of Λ^+ into $\mathbb{R}$ so that order statistics of Λ^+ valued variables are well defined.

B.1 THE ABSTENTION OPTION

This appendix expands the convention introduced in Section 3.3. The enlarged grid is $\Lambda^+ = \Lambda \cup \{\lambda_\dagger\}$ with $\lambda_\dagger \succ \lambda_K$, and $\lambda_\dagger$ is treated as a configuration like any other: the runtime is evaluated at it, it is eligible for the certified set, and it carries both a loss and a utility.

Consistency with the loss. Definition 1 charges an execution the average verifier exposure $|\mathcal{D}(\xi)|^{-1} \sum_{(t,m)\in\mathcal{D}(\xi)} (1 - q_t^m)$ of the proposals it deploys, and charges the maximal loss to an execution that exhausts its reflection budget. At $\lambda_\dagger$ the runtime deploys nothing and no execution is started, so $\mathcal{D}(\xi(\lambda_\dagger)) = \emptyset$ and there is no exposure to charge; the value $\ell(\xi(\lambda_\dagger)) = 0$ is the extension of the definition to that case rather than an additional stipulation. This is also why $\lambda_\dagger$ must be distinguished from an aborted execution, which does start and is charged the maximal loss: abstention is a decision not to deploy, an abort is a failure of a deployment that was attempted.

Accounting on the utility axis. The cost of abstaining is charged in full through $U(\lambda_\dagger) = 0$, so the two axes remain separated: the risk axis records the exposure a deployment creates and the utility axis records what a deployment achieves. A certificate that returns $\lambda_\dagger$ therefore reports a vacuous risk and a zero utility simultaneously, and cannot be mistaken for a favorable outcome. In Section 4.3 the option is present at every target level and is omitted from the reported certified set, so that the set column lists only deployable configurations.

Effect on the majorant. Because $\lambda_\dagger$ is the largest element of Λ^+ and $L_i(\lambda_\dagger) = 0$, including it in the running maximum $\widetilde{L}_i(\lambda) = \max_{\lambda' \succeq \lambda} L_i(\lambda')$ never raises the majorant at any $\lambda \in \Lambda$, and $\widetilde{L}_i(\lambda_\dagger) = 0$. The abstention option therefore supplies feasibility at the top of the grid at no cost in the gap $\widetilde{R}_n - \widehat{R}_n$, and the gap continues to measure only the non monotonicity of L_i on the deployable candidates.

Why the convention is not vacuous. Adding $\lambda_\dagger$ makes $\widehat{\mathcal{F}}(\alpha)$ non empty at every level, so non emptiness carries no information and the substance of a certificate moves to two reported quantities: whether $\widehat{\lambda}(\alpha) \in \Lambda$, and the utility attained there. The alternative designs are less honest rather than more informative. Assuming (C3) outright asserts $L_i(\lambda_K) \le \alpha$ almost surely, which fails for every $\alpha < 1$ as soon as the most conservative candidate aborts with positive probability; returning λ_K whenever the certified set is empty reports a configuration whose own calibration bound exceeds the target, as happens at $\alpha = 0.10$ in Table 2b. Reporting an abstention instead states exactly what the calibration data support, namely that no deployable configuration is certifiable at that level with n instances. This is the standard reading of a reject option in selective prediction (Chow, 1970; Geifman & El-Yaniv, 2017), transposed from a per instance decision to a decision about the deployed configuration: the object declined here is not a single prediction but the deployment itself.

B.2 DEFERRED STATEMENT

Proposition 1 (Policy optimization improves the certificate)**.** *Let the policy be optimized on the training split and let the certificate be computed on an independent calibration split of size n, so that Theorems 1 and 2 apply to each policy. Suppose the optimized runtime dominates the original one pathwise, that is, there is a coupling of the two calibration samples under which*

$$\widetilde{L}_i^{\mathrm{new}}(\lambda) \;\le\; \widetilde{L}_i^{\mathrm{old}}(\lambda) \qquad \text{for every } \lambda \in \Lambda^+ \text{ and every } i \le n, \text{ almost surely.}$$

Then, for every $\lambda \in \Lambda^+$, every α and every δ admissible in Theorem 2: (i) *the certified level decreases stochastically, $\widehat{\alpha}_\delta^{\mathrm{new}}(\lambda) \preceq_{\mathrm{st}} \widehat{\alpha}_\delta^{\mathrm{old}}(\lambda)$;* (ii) *the certified set expands stochastically, $\lambda^\star_{\delta,\mathrm{new}}(\alpha) \preceq_{\mathrm{st}} \lambda^\star_{\delta,\mathrm{old}}(\alpha)$;* (iii) *the CRC certified sets nest and the selection becomes no more conservative, $\widehat{\mathcal{F}}_{\mathrm{old}}(\alpha) \subseteq \widehat{\mathcal{F}}_{\mathrm{new}}(\alpha)$ and $\widehat{\lambda}_{\mathrm{new}}(\alpha) \le \widehat{\lambda}_{\mathrm{old}}(\alpha)$ under the coupling.*

The hypothesis is the functional form of the statement that the optimized runtime is uniformly better across configurations. It is stronger than a comparison of population means or aggregate certified bounds at each λ. The per configuration comparison of Section 4.4 reports aggregate empirical evidence only and does not verify this pathwise hypothesis.

B.3 Proof of Lemma 1

Proof. Write $\theta = \tau(\lambda)$ and $\theta' = \tau(\lambda')$; since τ is non decreasing, $\lambda \le \lambda'$ gives $\theta \le \theta'$. By construction the reflection sequence at s is generated from $(s, z^{(j)}, e^{(j)})$ alone and therefore does not depend on the configuration; the configuration determines only the round at which the loop stops. Since $\{j \le B : q^{(j)} \ge \theta'\} \subseteq \{j \le B : q^{(j)} \ge \theta\}$, the minimum over the smaller set is not smaller, so $j(\lambda) \le j(\lambda')$; in particular $j(\lambda') < \infty$ forces the smaller set to be non empty and hence $j(\lambda) < \infty$.

If $q^{(j(\lambda))} \ge \theta'$, then $j(\lambda)$ belongs to the smaller set, so $j(\lambda') \le j(\lambda)$; combined with the previous inequality, $j(\lambda') = j(\lambda)$ and the two deployed scores coincide. Otherwise $q^{(j(\lambda))} < \theta'$, so $j(\lambda)$ does not belong to the smaller set and hence $j(\lambda') > j(\lambda)$; by definition of $j(\lambda')$ we then have $q^{(j(\lambda'))} \ge \theta' > q^{(j(\lambda))}$. In both cases $q^{(j(\lambda'))} \ge q^{(j(\lambda))}$. □

B.4 Proof of Theorem 1

Proof. We verify the three conditions of Section 2.2 for the family $\{\widetilde{L}_i(\cdot)\}_{i=1}^{n+1}$ and then invoke conformal risk control (Angelopoulos et al., 2022).

(C1) Exchangeability. The map $(x, \omega) \mapsto \widetilde{L}(\cdot)$ is a fixed measurable function, being the composition of $\lambda \mapsto \ell(\Xi(x, \omega; \lambda))$ with the running maximum over the finite set Λ^+. Applying one fixed map coordinatewise to the exchangeable sequence $(x_i, \omega_i)_{i=1}^{n+1}$ preserves exchangeability.

(C2) Boundedness and monotonicity. $\ell \in [0, 1]$ by Definition 1, so $\widetilde{L}_i \in [0, 1]$. For $\lambda \le \lambda'$ in Λ^+ the maximum defining $\widetilde{L}_i(\lambda')$ is taken over a subset of the set defining $\widetilde{L}_i(\lambda)$, whence $\widetilde{L}_i(\lambda') \le \widetilde{L}_i(\lambda)$. Monotonicity therefore holds for every realization by construction rather than by assumption. Right continuity is vacuous on a finite grid.

(C3) Feasibility at the most conservative candidate. The largest element of Λ^+ is $\lambda_\dagger$, and $\widetilde{L}_i(\lambda_\dagger) = L_i(\lambda_\dagger) = 0 \le \alpha$ almost surely by the abstention convention of Section 3.3. In particular $\frac{n}{n+1}\widetilde{R}_n(\lambda_\dagger) + \frac{1}{n+1} = \frac{1}{n+1} \le \alpha$, so $\lambda_\dagger \in \widehat{\mathcal{F}}(\alpha)$ and the set is non empty; $\widehat{\lambda}(\alpha)$ is therefore well defined.

Conformal risk control applied to $\{\widetilde{L}_i\}$ at level α with the selection $\widehat{\lambda}(\alpha) = \min \widehat{\mathcal{F}}(\alpha)$ gives $\mathbb{E}[\widetilde{L}_{n+1}(\widehat{\lambda}(\alpha))] \le \alpha$.

It remains to extend this to the whole certified set. Since each $\widetilde{L}_i$ is non increasing, so is $\widetilde{R}_n$, hence $\lambda \in \widehat{\mathcal{F}}(\alpha)$ and $\lambda' \ge \lambda$ imply $\lambda' \in \widehat{\mathcal{F}}(\alpha)$; the set is upward closed and equals $\{\lambda \in \Lambda^+ : \lambda \ge \widehat{\lambda}(\alpha)\}$. Fix $\lambda \in \widehat{\mathcal{F}}(\alpha)$. Then $\lambda \ge \widehat{\lambda}(\alpha)$ holds for every realization of the calibration sample, and since $\widetilde{L}_{n+1}$ is non increasing for every realization,

$$L_{n+1}(\lambda) \;\le\; \widetilde{L}_{n+1}(\lambda) \;\le\; \widetilde{L}_{n+1}\big(\widehat{\lambda}(\alpha)\big) \qquad \text{pointwise.}$$

Taking expectations and applying the display above yields $\mathbb{E}[L_{n+1}(\lambda)] \le \alpha$. □

B.5 Proof of Theorem 2

We first record the exchangeability argument in the form we use.

Lemma 2 (Rank bound)**.** *Let $V_1, \ldots, V_{n+1}$ be exchangeable real valued random variables, let $k \in \{1, \ldots, n\}$, and let $V_{(k)}$ denote the k-th smallest of $V_1, \ldots, V_n$. Then* $\Pr[V_{n+1} \le V_{(k)}] \ge k/(n+1)$.

Proof. For $j \le n+1$ let $\rho_j = 1 + |\{i \le n+1 : V_i < V_j\}|$ be the rank of V_j. Sorting $V'_{(1)} \le \cdots \le V'_{(n+1)}$ over all $n+1$ variables, the value at sorted position p has strictly fewer than p values below it, so $\rho_j \ge k+1$ forces that value to occupy a sorted position $p \ge k+1$. Hence

$\sum_{j\le n+1}\mathbf{1}\{\rho_j \ge k+1\} \le n+1-k$ surely. Taking expectations and using that exchangeability makes $\Pr[\rho_j \ge k+1]$ identical for all j gives $\Pr[\rho_{n+1} \ge k+1] \le (n+1-k)/(n+1)$.

If $V_{n+1} > V_{(k)}$ then at least k of $V_1, \dots, V_n$ are at most $V_{(k)}$ and hence strictly below V_{n+1}, so $\rho_{n+1} \ge k+1$. Therefore $\Pr[V_{n+1} > V_{(k)}] \le (n+1-k)/(n+1)$, which is the claim. □

Proof of Theorem 2. Fix α and put $V_i = \widehat{\lambda}_i(\alpha)$ for $i \le n+1$. The defining set $\{\lambda \in \Lambda^+ : \widetilde{L}_i(\lambda) \le \alpha\}$ contains $\lambda_\dagger$, since $\widetilde{L}_i(\lambda_\dagger) = 0 \le \alpha$, so V_i is a well defined element of Λ^+ and the minimum is attained. As in the proof of Theorem 1, V_i is the image of (x_i, ω_i) under one fixed measurable map, so $V_1, \dots, V_{n+1}$ are exchangeable.

Let $k = \lceil (1-\delta)(n+1) \rceil$; the hypothesis $\delta \ge 1/(n+1)$ gives $(1-\delta)(n+1) \le n$ and hence $k \le n$, so Lemma 2 applies to $\lambda^\star_\delta(\alpha) = V_{(k)}$ and yields

$$\Pr\big[\widehat{\lambda}_{n+1}(\alpha) \le \lambda^\star_\delta(\alpha)\big] \;\ge\; \frac{k}{n+1} \;\ge\; 1-\delta.$$

On that event, fix any $\lambda \in \Lambda^+$ with $\lambda \ge \lambda^\star_\delta(\alpha)$. Then $\lambda \ge \widehat{\lambda}_{n+1}(\alpha)$, and since the minimum defining $\widehat{\lambda}_{n+1}(\alpha)$ is attained we have $\widetilde{L}_{n+1}(\widehat{\lambda}_{n+1}(\alpha)) \le \alpha$; monotonicity of $\widetilde{L}_{n+1}$ then gives $\widetilde{L}_{n+1}(\lambda) \le \alpha$, and $L_{n+1}(\lambda) \le \widetilde{L}_{n+1}(\lambda)$ completes the argument. The event does not depend on λ, so the conclusion holds simultaneously for all $\lambda \ge \lambda^\star_\delta(\alpha)$ on a single event of probability at least $1-\delta$. □

B.6 Proof of Corollary 1

Proof. Monotonicity of $\widehat{\alpha}_\delta$ is immediate: for $\lambda \le \lambda'$ we have $\widetilde{L}_i(\lambda) \ge \widetilde{L}_i(\lambda')$ for every i, and the k-th order statistic is non decreasing in each of its arguments.

For the identity, note first that for every i and every $\lambda \in \Lambda^+$,

$$\widetilde{L}_i(\lambda) \le \alpha \quad\Longleftrightarrow\quad \lambda \ge \widehat{\lambda}_i(\alpha),$$

since the implication from right to left follows from monotonicity together with $\widetilde{L}_i(\widehat{\lambda}_i(\alpha)) \le \alpha$, and the converse holds because λ then lies in the set whose minimum is $\widehat{\lambda}_i(\alpha)$. Consequently

$$\widehat{\alpha}_\delta(\lambda) \le \alpha \;\Leftrightarrow\; \big|\{i \le n : \widetilde{L}_i(\lambda) \le \alpha\}\big| \ge k \;\Leftrightarrow\; \big|\{i \le n : \widehat{\lambda}_i(\alpha) \le \lambda\}\big| \ge k \;\Leftrightarrow\; \lambda^\star_\delta(\alpha) \le \lambda,$$

where the first and last equivalences are the definitions of the k-th order statistic of $\{\widetilde{L}_i(\lambda)\}_{i\le n}$ and of $\{\widehat{\lambda}_i(\alpha)\}_{i\le n}$ respectively. □

B.7 Proof of Proposition 1

Proof. Because the policy is a measurable function of the training split alone and the calibration split is independent of it, conditionally on the training split the calibration instances of the optimized runtime are exchangeable and the losses are bounded. Theorems 1 and 2 therefore apply to each policy, with all probabilities understood conditionally on the training split. This is the only role of data splitting, and it is necessary: computing the certificate on data used for optimization would make the calibration losses dependent on the fitted policy and violate exchangeability.

Work under the stated coupling, so that $\widetilde{L}_i^{\mathrm{new}}(\lambda) \le \widetilde{L}_i^{\mathrm{old}}(\lambda)$ for all $i \le n$ and all λ almost surely.

(i) Fix λ. The k-th order statistic is non decreasing in each argument, so $\widehat{\alpha}^{\mathrm{new}}_\delta(\lambda) \le \widehat{\alpha}^{\mathrm{old}}_\delta(\lambda)$ almost surely under the coupling, which is exactly stochastic dominance for the two marginal laws.

(ii) Fix α. Pointwise domination gives $\{\lambda : \widetilde{L}_i^{\mathrm{old}}(\lambda) \le \alpha\} \subseteq \{\lambda : \widetilde{L}_i^{\mathrm{new}}(\lambda) \le \alpha\}$, hence $\widehat{\lambda}_i^{\mathrm{new}}(\alpha) \le \widehat{\lambda}_i^{\mathrm{old}}(\alpha)$ for every i. Applying the k-th order statistic, which is again monotone in each argument, gives $\lambda^\star_{\delta,\mathrm{new}}(\alpha) \le \lambda^\star_{\delta,\mathrm{old}}(\alpha)$ almost surely.

(iii) Averaging the pointwise domination gives $\widetilde{R}_n^{\mathrm{new}}(\lambda) \le \widetilde{R}_n^{\mathrm{old}}(\lambda)$ for every λ, so the defining inequality of $\widehat{\mathcal{F}}(\alpha)$ is satisfied by the new policy whenever it is satisfied by the old one. Hence $\widehat{\mathcal{F}}_{\mathrm{old}}(\alpha) \subseteq \widehat{\mathcal{F}}_{\mathrm{new}}(\alpha)$, and taking minima gives $\widehat{\lambda}_{\mathrm{new}}(\alpha) \le \widehat{\lambda}_{\mathrm{old}}(\alpha)$. □

C TECHNICAL DETAILS

Verifier Subagent. CertAgent implements the verifier as a lightweight runtime subagent inside the Agent Runtime. The verifier scores the proposal using only local runtime evidence, including the current step number, previous observations, previous code actions, previous errors, and available attachments. At the reasoning stage, it evaluates two failure modes: *Control Error* and *Reasoning Insufficiency*. At the action stage, it evaluates *Argument Error* and *Execution Error*, checking for issues such as malformed Python code, missing tool calls, repeated actions, risky runtime surfaces, and unresolved prior failures. The final verifier score is the minimum of the component scores, clipped to a fixed numerical range $[q_{\min}, q_{\max}]$, and the verifier also returns structured feedback and a dominant failure mode for later reflection. The clipping range constrains the candidate grid: a configuration with $\tau(\lambda) > q_{\max}$ rejects every proposal and aborts every execution, so Λ is placed inside $[0, q_{\max}]$.

Gate Module. The gate module converts the global runtime control parameter λ into an acceptance threshold $\tau(\lambda)$. In our implementation, both the reasoning gate and the action gate use the same default identity mapping, i.e., $\tau(\lambda) = \lambda$ after clipping λ to $[0, 1]$. A proposal with verifier score q is accepted only when $q \geq \tau(\lambda)$. Otherwise, the runtime blocks the proposal before deployment and invokes the stage-specific reflection operator. The revised proposal is then scored by the same verifier and passed through the gate again. For each stage at each step, the initial proposal is round $j = 0$ and at most B reflection revisions, indexed by $j = 1, \ldots, B$, are allowed. If all proposals at rounds $j = 0, \ldots, B$ are rejected, the execution terminates, and Definition 1 charges the maximal system loss to that outcome. This design makes λ a single knob controlling the strictness of both reasoning-level and action-level certification.

Reflection. When a proposal is rejected by the gate, CertAgent invokes a stage-specific reflection operator. The reasoning reflector constructs a feedback prompt containing the rejected transition, the verifier score requirement, the verifier's failure-mode diagnosis, an excerpt of the rejected proposal, and the remaining reflection budget for the current stage and step. It asks the model to revise the reasoning transition by continuing to plan, taking a better-supported action, or finishing only when sufficient evidence is available. The action reflector follows the same structure but focuses on executable code, asking the model to preserve the intended direction while fixing missing arguments, malformed tool calls, or likely execution failures. Each reflection is appended to the model context as an additional user message, and the regenerated proposal is re-scored by the verifier before it can proceed. Because the prompt reports the current score requirement, the generated proposal sequence depends weakly on λ and the hypothesis of Lemma 1 is not satisfied exactly; removing the score requirement from the prompt would satisfy it. The certificate does not rely on this, since Theorems 1 and 2 are stated for the non increasing majorant.

Diagnostics for the calibration conditions. Three quantities computed on the calibration split make the conditions of Section 3.3 auditable. The *abort rate* at candidate λ is $n^{-1} \sum_{i \leq n} \mathbf{1}\{\xi_i(\lambda) \text{ aborts}\}$; a grid on which this rate is non negligible at the conservative end lies outside the regime where the majorant is useful. The *monotone path fraction* is $n^{-1}|\{i \leq n : L_i(\lambda_1) \geq \cdots \geq L_i(\lambda_K)\}|$, the fraction of calibration instances whose realized loss path is already non increasing; it is a direct and falsifiable check on condition (C2), which the majorant otherwise enforces at a cost. That cost is the *majorant gap* $\widetilde{R}_n(\lambda) - \widehat{R}_n(\lambda) \geq 0$, the amount by which the certified bound is loosened by not assuming (C2). Table 4 reports these quantities on the calibration split used for RQ2.

For the values in Table 4, $56/60 = 0.933$ of the calibration loss paths are already non increasing on the tested grid. The abort counts remain small, and the majorant gaps quantify the modest additional conservativeness introduced by enforcing condition (C2). The raw risks also satisfy the gate-implied consistency bound

$$\widehat{R}_n(\lambda) \leq 1 - \lambda + a_n(\lambda)\lambda,$$

where $a_n(\lambda)$ is the empirical abort rate. For example, at $\lambda = 0.90$ the bound is $0.10 + (2/60) \times 0.90 = 0.1300$, above the reported raw risk 0.1203.

Table 4: Calibration diagnostics and single-deployment certificates for **RQ2**. The calibration size is $n = 60$. Utility is the task success fraction, with exact solved/evaluated counts shown; decimal values are rounded to four places. The CRC level is $\frac{n}{n+1}\widetilde{R}_n(\lambda) + \frac{1}{n+1}$, and $\widehat{\alpha}_{0.1}(\lambda)$ is the 55th smallest majorized calibration loss, as required by Theorem 2 for $\delta = 0.1$.

λ	Solved/evaluated	Utility	$\widehat{R}_n$	$\widetilde{R}_n$	Majorant gap	Aborts/evaluated	CRC level	$\widehat{\alpha}_{0.1}(\lambda)$
0.10	11/60	0.1833	0.8458	0.8525	0.0067	0/60 (0.0000)	0.8549	0.8500
0.30	13/60	0.2167	0.6467	0.6600	0.0133	0/60 (0.0000)	0.6656	0.6500
0.50	14/60	0.2333	0.4508	0.4692	0.0183	0/60 (0.0000)	0.4779	0.4500
0.70	13/60	0.2167	0.2625	0.2750	0.0125	1/60 (0.0167)	0.2869	0.2500
0.90	11/60	0.1833	0.1203	0.1203	0.0000	2/60 (0.0333)	0.1348	0.0900

D Reproducibility

D.1 Large Language Models

For open source large language models, we use the official implementations provided by the Hugging Face `transformers` library for model loading and inference. All open source models are evaluated under the same inference interface to ensure consistent decoding and execution settings across experiments.

For closed source models, we use the official OpenAI API. Specifically, GPT-5.5 is accessed through the OpenAI API with the reasoning effort set to `xHigh`. Unless otherwise specified, all model outputs are generated under the same task prompts and agent runtime protocol used in the main experiments.

D.2 Baseline Agents

For baseline agent systems, including SMOLAGENTS and AUTOAGENT, we use the official implementations released in their respective repositories. We keep the original agent designs and default orchestration logic whenever possible, and only adapt the input and output interfaces required by our evaluation pipeline.

To ensure a fair comparison across different agent systems, all agents are evaluated with the same Live-SWE harness. This ensures that each method is given the same problem solving interface, execution environment, and evaluation protocol, so that performance differences mainly reflect the agent runtime design rather than differences in harness capability.

D.3 Benchmark Details

We conduct our main experiments on SWE-Bench Lite, a widely used benchmark for evaluating software engineering agents on real world GitHub issues. SWE-Bench Lite provides a compact and practical evaluation setting while preserving the core requirement of resolving repository level programming tasks.

We split SWE-Bench Lite into training, calibration, and test sets with a ratio of 0.6, 0.2, and 0.2, respectively, giving a calibration set of $n = 60$ instances. The training split is used for policy optimization, the calibration split is used for CertAgent risk certification, and the test split is used only for final evaluation. This separation ensures that the certification procedure is calibrated independently from both policy training and final test evaluation, which is the condition invoked in the proof of Proposition 1.

D.4 Policy Optimization

For the policy optimization experiments, we fine tune the open source backbone models through the Hugging Face training pipeline. We adopt LoRA based parameter efficient fine tuning to reduce memory and computational cost while keeping the backbone model parameters frozen. The LoRA adapters are applied to the attention and feed forward projection layers, including `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`.

The optimization follows the risk penalized objective introduced in Section 3.4, where the policy is optimized to improve task utility while reducing system level Agent Runtime risk. We set the per-stage, per-step reflection budget to $B = 24$, use $\lambda = 0.5$ as the runtime configuration, and set the risk penalty coefficient to $\beta = 0.25$. We train for 10 epochs over the training split and perform one policy update after each training instance. The learning rate is set to 1.0×10^{-6}, the weight decay is set to 0, the baseline momentum is set to 0.9, and the maximum gradient norm is clipped to 1.0.

For LoRA, we use rank $r = 8$, scaling factor $\alpha = 32$, and dropout rate 0.05. During generation, we use a maximum generation length of 192 tokens with sampling enabled, temperature 0.7, and top-p 0.95. Gradient checkpointing is enabled to reduce memory usage. We save both the trained LoRA adapter and the merged model for downstream evaluation.

D.5 Computing Resources

All experiments are conducted on a workstation equipped with an AMD EPYC 7763 64-Core CPU and NVIDIA RTX Pro 6000 Blackwell Workstation Edition GPUs. The same computing environment is used across baseline agents and CertAgent whenever applicable, in order to minimize implementation dependent variation in runtime and efficiency measurements.