UNDERSTANDING EFFECTIVENESS, EFFICIENCY, AND IDENTIFIABILITY IN AGENTIC EVOLUTION: A CONTROL-THEORETIC PERSPECTIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic evolution improves agent systems through iterative evaluation and patching, yet existing work primarily focuses on performance, leaving the dynamics of the evolution process insufficiently understood. We present a control-theoretic framework for understanding agentic evolution as a discrete-time closed-loop system, which analyzes three properties that directly characterize the evolution process, including: *(i) Effective* evolution favors a smaller patch radius with stronger local error contraction, *(ii) Efficient* evolution benefits from lower initial error, faster contraction, and a smaller patch-to-error ratio, and *(iii) Identifiable* evolution requires stronger observable patch effects, lower effect coherence, and fewer simultaneously active patch directions. Experiments on three mainstream agentic benchmarks validate these theoretical findings. Guided by these findings, we propose TRUSTEVO, a trust-region framework that explores failure-driven patch directions and adaptively constrains patch size and component-edit scope according to observed outcomes. TRUSTEVO outperforms the strongest baseline by 3.9 percentage points on average while using 16.5% fewer evolution tokens overall, demonstrating its effectiveness and efficiency.

1 INTRODUCTION

Agent evolution has emerged as a practical approach to improving the capabilities of agent systems (ang Gao et al., 2026). A typical evolution loop repeatedly executes an agent system on task data or in the environment, asks a patch generator to generate a patch, and applies that patch to the agent system or the corresponding configuration so that errors decrease over successive epochs.

Most existing agentic evolution work emphasizes the performance of the agent system after evolution rather than the dynamics of the evolution. This leaves three key aspects of agentic evolution insufficiently understood, limiting the further improvement of future research, including: *(i)* ***Evolution Effectiveness***: which factors determine the agentic system error after evolution; *(ii)* ***Evolution Efficiency***: how to quantify the cost during agentic evolution; *(iii)* ***Evolution Identifiability***: how to quantify the ambiguity in identifying the optimal evolving directions throughout evolution.

To better understand the dynamics of agentic evolution and thereby inspire future improvements, in this paper, ***we present an analysis framework for agent evolution through a control-theoretic formulation***. Unlike prior work that uses control theory solely to model agent systems (Eslami & Yu, 2026; Nosrati et al., 2026; Prinos et al., 2026), we leverage control theory for a fine-grained analysis of how different factors affect distinct aspects of agent evolution. We represent the agentic evolution process as a discrete-time closed-loop system. The overall agent evolution process consists of two nested components: the agent system generates rollouts that are provided to the patch generator, and the patch generator uses the rollouts to modify the agent system, improving its performance.

Our analysis framework offers three benefits: *(i) Comprehensive Process Analysis*: we extend the analysis from final evolution error to evolution cost and ambiguity, inspiring improvements from various aspects of agentic evolution. *(ii) System Independent Generality*: our framework is independent of any particular agent architecture, planner, or component type, showing high generalization. *(iii) Executable Design Insights*: we reveal how different factors shape the evolution process, shedding light on future improvements in agentic evolution.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

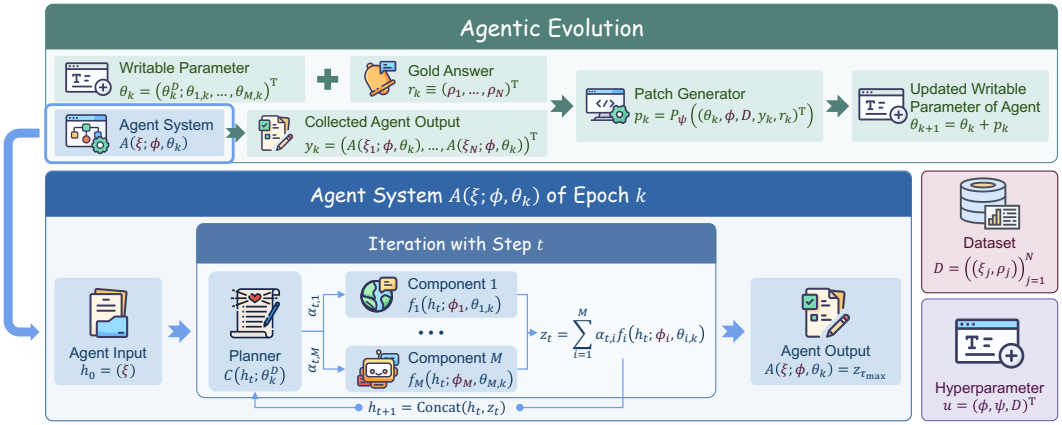


Figure 1: The illustration of our analysis framework of agentic evolution **to better understand the meaning of mathematical symbols in §2**. We represent agentic evolution as a closed-loop process that repeatedly optimizes the agent system based on the agent output of each epoch. At each epoch, the agent system generates output based on the given data. A patch generator then consumes the current agent and system output to generate a patch for the writable parameters of the agent system. Dataset and hyperparameters are fixed during the evolution.

Analysis based on our framework identifies the key factors governing the above three aspects of agent evolution, including: *(i) Evolution Error* depends on patch size and feedback size, motivating smaller and more accurate updates. *(ii) Evolution Cost* is mainly affected by initial error and patch size, suggesting better initialization and smaller contraction. *(iii) Evolution Ambiguity* is determined by how well evaluation signals distinguish writable parameter directions, which can be reduced with a smaller feedback size and less overlap among component-edit directions.

To shed light on future agentic evolution research, based on the above analysis, we present TRUSTEVO, a controlled patch-based agentic evolution framework based on the trust region. TRUSTEVO identifies likely failure directions from evaluation feedback, generates patches within the current trust region, and accepts them only when they yield reliable gains without unacceptable regressions. The trust region is then adapted according to previous patch outcomes. Across three challenging benchmarks, TRUSTEVO outperforms the strongest baseline by 3.9 on average with 16.5% fewer evolution tokens, showing the effectiveness and efficiency of TRUSTEVO.

Our main contributions are as follows:

- We present an analysis framework based on control theory to inspire future research on agentic evolution, which can provide comprehensive process analysis, system independent generality, and design insights into the agentic evolution process.
- Based on our framework, we derive factor-level characterizations showing how different factors affect the effectiveness, efficiency, and identifiability of agentic evolution.
- We propose TRUSTEVO, a trust-region agentic evolution based on our analysis, which demonstrates effectiveness and efficiency on three mainstream agentic benchmarks, with an average improvement of 3.9 using 16.5% fewer tokens.

2 ANALYSIS FRAMEWORK OF AGENTIC EVOLUTION

In this section, we present our analysis framework of agentic evolution based on control theory, which is illustrated in Figure 1. We represent agentic evolution as a closed-loop process that repeatedly optimizes agent parameters using a patch generator over multiple epochs. At each epoch, the agent system first produces output based on the given data. A patch generator then consumes the current agent and output to generate a patch for the writable parameters of the agent system. We model the closed-loop process described above as an automatic control system for analysis.

2.1 AGENT SYSTEM FORMALIZATION

We consider an agent system composed of a planner and M components (e.g., RAG, memory, or skill modules). Given a user input, the agent executes a finite number of steps and returns a final output. At each step, the planner assigns component weights based on the session history, where unused components are assigned zeros. Then, each component produces an output based on the session history, and the weighted sum of these outputs becomes the output of the current step, which is used as the input for the next step. This abstraction does not assume a specific architecture or component type, ensuring generalization. In Appendix C, we provide several examples illustrating how the above formalization can characterize mainstream agent systems, demonstrating the generalization. For simplicity, let $\tau_{\max}$ denote the maximum number of steps across all data. For data that terminate before reaching $\tau_{\max}$, we can pad the remaining steps by having the agent directly copy the input to the output at each step after completing its reasoning.

The values of the writable parameters of the agentic system at evolution epoch k are

$$\theta_k = (\theta_k^D, \theta_{1,k}, \ldots, \theta_{M,k})^\top, \tag{1}$$

where $\theta_k^D \in \mathbb{R}^{n_D}$ parameterizes the conditional planner and $\theta_{i,k} \in \mathbb{R}^{n_i}$ parameterizes component i. Each component also has a fixed channel

$$\phi = (\phi_1, \ldots, \phi_M)^\top,\ \phi_i \in \mathbb{R}^{a_i}. \tag{2}$$

The fixed channel constrains the role of each component during evolution, which is independent and may have different dimensions across different components. For example, it can prevent a memory module from being repurposed as a skill store during evolution.

For an input $\xi \in \mathbb{R}^d$, let $z_0 = \xi$. We record the session history and zero-pad unused positions so that all histories have the same dimension:

$$h_t = (z_0, z_1, \ldots, z_{t-1}, 0, \ldots, 0)^\top \tag{3}$$

where $z_t \in \mathbb{R}^d$ is the step-t agent output. The planner is a conditional generator that returns the probability weight vector:

$$\alpha_t = C(h_t; \theta_k^D) \in \mathbb{R}^M,\ \ \alpha_t \geq 0,\ \mathbf{1}^\top \alpha_t = 1. \tag{4}$$

Component i is a conditional predictor f_i with component output and aggregate output

$$z_t = \sum_{i=1}^{M} \alpha_{t,i} f_i(h_t; \phi_i, \theta_{i,k}) \in \mathbb{R}^d. \tag{5}$$

After $\tau_{\max}$ steps, the agent system's prediction for one input of the k-th epoch is:

$$A(\xi; \phi, \theta_k) = z_{\tau_{\max}} \in \mathbb{R}^d. \tag{6}$$

2.2 EVOLUTION FORMALIZATION

Based on the agent system, evolution uses a fixed patch generator to reduce the agent's error on a dataset that is human-labeled or collected through environmental interaction. Write one instance as $(\xi, \rho) \in \mathbb{R}^d \times \mathbb{R}^d$, where ξ is the input and ρ is the gold answer, and define the dataset for the evolution run as:

$$D = ((\xi_j, \rho_j))_{j=1}^N. \tag{7}$$

Because changing D can change the convergence point of the system, the dataset is fixed during the evolution. Online evolution can be interpreted as repeatedly running the evolution to stability after each new data point is introduced, which can also be analyzed by our framework.

The controlled state contains only writable parameters θ_k. The hyperparameters remain fixed during evolution:

$$u_k \equiv u = (\psi, \phi, D)^\top, \qquad k = 0, 1, 2, \ldots \tag{8}$$

where ψ parameterizes the patch generator P_ψ. Once the values of hyperparameters are determined, the convergence result of the agentic evolution is fixed.

Table 1: The main findings of our framework that how different factors affect three properties of the agentic evolution. ↑ / ↓ denotes larger / smaller is better.

Property	**Favorable Factors**
Effectiveness ($\|e_\infty\| \downarrow$)	Patch radius $\overline{\Delta} \downarrow$ Contraction factor $\eta \downarrow$
Efficiency ($\mathcal{E}_\infty \downarrow$)	Initial error $\|e_0\| \downarrow$ Contraction rate $q \downarrow$ Patch-to-error ratio $c \downarrow$
Identifiability ($\lambda_{\min}(W_{\text{patch}}) \uparrow$)	Observable patch effect $\alpha \uparrow$ Effect coherence $\mu \downarrow$ Active directions $m \downarrow$

For each epoch k, let the agent output and the gold answer be:

$$y_k = \mathcal{Y}(D; \phi, \theta_k) = \begin{bmatrix} A(\xi_1; \phi, \theta_k) \\ \vdots \\ A(\xi_N; \phi, \theta_k) \end{bmatrix}, \; r_k \equiv \Gamma(D) = \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_N \end{bmatrix}. \tag{9}$$

The patch p_k modifies only writable parameters:

$$p_k = P_\psi \left(\begin{bmatrix} \theta_k \\ \phi \\ D \\ y_k \\ r_k \end{bmatrix} \right) = \begin{bmatrix} p_k^D \\ p_{1,k} \\ \vdots \\ p_{M,k} \end{bmatrix}, \; \theta_{k+1} = \theta_k + p_k \tag{10}$$

where p_k^D updates the planner and $p_{i,k}$ updates component i. The overall optimization objective is to use the patch generator to modify the writable parameters θ_k of the agent system so that the agent output y_k is as close as possible to the gold answer r_k.

3 Analysis of Effectiveness, Efficiency, and Identifiability

Based on the closed-loop formalization above, we study three properties that can directly guide the design of agentic evolution: *(i) Effectiveness*: the coverage error after evolution; *(ii) Efficiency*: the cost incurred during evolution; and *(iii) Identifiability*: whether observed failures can distinguish between different candidate patch directions. The main findings are summarized in Table 1. All proofs of this section are shown in Appendix A. We discuss further properties of agentic evolution in Appendix B to further demonstrate the power of our framework.

For fixed hyperparameters $u = [\psi; \phi; D]$, define the error map on the dataset and its Jacobian as

$$e(\theta) = \mathcal{Y}(D; \theta, \phi) - \Gamma(D), \qquad J(\theta) = \frac{\partial e(\theta)}{\partial \theta}. \tag{11}$$

In what follows, we suppress the dependence on the fixed hyperparameter u, write $e_k = e(\theta_k)$ and $p_k = p(\theta_k)$, and use the Euclidean norm and its induced operator norm throughout.

3.1 Evolution Effectiveness

We first study the effectiveness of agentic evolution, which is measured by the prediction error $e(\theta)$:

Theorem 1 (Upper Bound on Evolution Effectiveness) *Let $\mathcal{K} \subset \mathbb{R}^n$ be a compact convex set that contains the evolution trajectory as well as every line segment between θ_k and $\theta_k + p_k$. Assume that J is L-Lipschitz on $\mathcal{K}$, i.e.,*

$$\|J(\theta_1) - J(\theta_2)\| \le L\|\theta_1 - \theta_2\|, \qquad \forall\, \theta_1, \theta_2 \in \mathcal{K}, \tag{12}$$

and that the generated patches provide a contraction:

$$\|e_k + J(\theta_k) p_k\| \le \eta \|e_k\|, \qquad 0 \le \eta < 1. \tag{13}$$

If the patch at every epoch k satisfies

$$\|p_k\| \leq \overline{\Delta}, \tag{14}$$

then

$$\limsup_{k\to\infty} \|e_k\| \leq \frac{L\overline{\Delta}^2}{2(1-\eta)}. \tag{15}$$

Theorem 1 characterizes two conditions required for highly effective evolution. A smaller patch radius $\overline{\Delta}$ suppresses the second-order error incurred when moving away from the neighborhood, while a smaller contraction factor η makes the intended update more effective. Therefore, larger modifications are not inherently more beneficial. Even when the patch directions are effective, the corresponding nonlinear remainder still grows quadratically with the patch radius. This motivates generating the smallest patch that resolves the current failure and expanding the modification scope only after localized updates have been repeatedly validated as effective.

3.2 Evolution Efficiency

We next study evolution efficiency, which we quantify as:

$$\mathcal{E}_K = \sum_{k=0}^{K-1} \left(\|e_k\|^2 + \|p_k\|^2\right). \tag{16}$$

This quantity jointly measures the residual task error and the magnitude of successive updates. It captures the cumulative magnitude of the error presented to the evolution process and the patches produced in response, thereby quantifying the overall cost of evolution.

Theorem 2 (Upper Bound on Evolution Efficiency) *Suppose that, for all $k \geq 0$, there exist constants $0 \leq q < 1$ and $c \geq 0$ such that*

$$\|e_{k+1}\| \leq q\|e_k\|, \tag{17}$$

and

$$\|p_k\|^2 \leq c\|e_k\|^2. \tag{18}$$

Then, we can derive that

$$\mathcal{E}_\infty \leq \frac{1+c}{1-q^2}\|e_0\|^2. \tag{19}$$

The evolution efficiency is controlled by three factors with clear interpretations: the error $\|e_0\|$ upon entering the evolution stage, the contraction rate q, and the patch-to-error ratio c. Better initialization can reduce the error before fine-grained evolution begins, while patches that induce stronger contraction can reduce the cost when the system remains far from a good solution.

3.3 Evolution Identifiability

The preceding analysis assumes that the evolution method can identify an effective patch direction. In practical agents, however, different prompts, tools, memory rules, or workflow stages may produce similar failure signatures. When multiple candidate patch directions have highly overlapping effects on the observed output, failure attribution becomes ambiguous, forcing the optimizer to explore a larger number of candidate directions to find the optimized patch direction.

To characterize this property, define:

$$V = [v_1, \ldots, v_m] \in \mathbb{R}^{n\times m} \tag{20}$$

to contain m normalized candidate patch directions selected around the current state θ, where $\|v_i\| = 1$. A direction may correspond to a component-specific patch, a local decision rule, or another writable degree of freedom available to the optimizer. Its effect on the fixed-data error is

$$g_i = J(\theta)v_i, \qquad G = [g_1, \ldots, g_m]. \tag{21}$$

We define the local identifiability Gramian as

$$W_{\text{patch}} = G^\top G. \tag{22}$$

Larger eigenvalues of W_{patch} indicate that different active patch directions induce stronger effects on the output and are easier to distinguish from one another.

For $m > 1$, define the coherence among the output effects of these directions as

$$\mu = \max_{i \neq j} \frac{|g_i^\top g_j|}{\|g_i\| \, \|g_j\|}, \tag{23}$$

and let $\mu = 0$ when $m = 1$.

Theorem 3 (Coherence Lower Bound for Evolution Identifiability) *Suppose that all patch directions satisfy*

$$\|g_i\| \geq \alpha > 0, \qquad i = 1, \ldots, m, \tag{24}$$

and

$$(m-1)\mu < 1. \tag{25}$$

Then

$$\lambda_{\min}(W_{\text{patch}}) \geq \alpha^2 \big(1 - (m-1)\mu\big). \tag{26}$$

Evolution identifiability is stronger when candidate patch directions produce stronger observable effects (larger α) and lower overlap among those effects (smaller μ). When nonzero overlap is present, increasing the number of simultaneously active directions weakens the guaranty on the minimum eigenvalue through the factor $1-(m-1)\mu$. Therefore, when multiple writable components produce correlated failure signatures, restricting each evolution epoch to a small subset of failure-relevant components can improve the conditioning of failure attribution. Additional observations are useful only when they increase either the strength or the distinguishability of these effects. Simply duplicating similar data does not improve the lower bound. Thus, compared with indiscriminately enlarging the feedback set, richer and more diverse execution evidence, intermediate-state checks, and component-specific failure signals are more effective for improving identifiability.

4 Validation Experiments

4.1 Experimental Setup

We evaluate our conclusion on agentic evolution in §3 using three mainstream agentic benchmarks, including τ^2-Bench (Barres et al., 2026), Terminal-Bench-Hard (Merrill et al., 2026), and WebArena-Verified-Hard (Zhou et al., 2023). We introduce such benchmarks in Appendix G. For each benchmark, we start from its official baseline agentic system and use GPT-5.4-NANO as the backbone model. Evolution is performed by CODEX based on GPT-5.4-MINI, which generates and applies a patch based on the current agent system and trajectories.

We detail how to modify the initial agent system to obtain different hyperparameters for different evolution processes in Appendix H. We also discuss how to calculate each property and factor used in experiments in Appendix D. Because practical systems contain many confounding factors, the experiments are intended to validate the qualitative trends derived in §3. Results are normalized independently within each benchmark to show the overall trend.

4.2 Validation Experiments

The validation experiment is shown in Figure 2, which provides complementary evidence supporting our theoretical findings. *(i) Effectiveness*: The final normalized residual error generally decreases as the normalized effectiveness bound becomes smaller. Despite moderate scatter across different settings, the overall monotonic trend is clear, with a benchmark-balanced R^2 of approximately 0.81 and a Spearman correlation of about 0.84. This indicates that the upper bound of Theorem 1 is a meaningful predictor of the eventual optimization accuracy. *(ii) Efficiency*: The result shows a similar relationship between the theoretical efficiency bound and the empirical evolution cost $\mathcal{E}_\infty$. Settings with smaller predicted costs tend to incur lower cumulative residual and update energy, yielding an R^2 of roughly 0.76 and a Spearman correlation of around 0.80. The remaining dispersion is expected because the empirical trajectories are affected by finite-step dynamics and setting-specific variations. *(iii) Identifiability*: The figure directly examines the identifiability lower bound. All evaluated settings lie on or above the reference line $Y = X$, confirming the lower bound of Theorem 3. Overall, experimental results consistently support the predicted links among findings in §3.

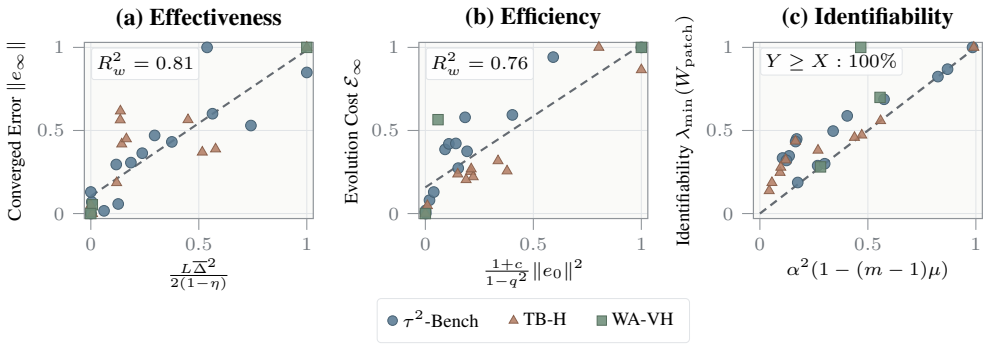


Figure 2: The validation experiments of our findings in §3. Different points denote different hyperparameters. The detailed calculation of each property and factor is discussed in Appendix D. R_w^2 denotes the weighted coefficient of determination, and $Y \geq X$ denotes how many results satisfy the relation in Theorem 3. Result under each setting is normalized to reflect the overall trend.

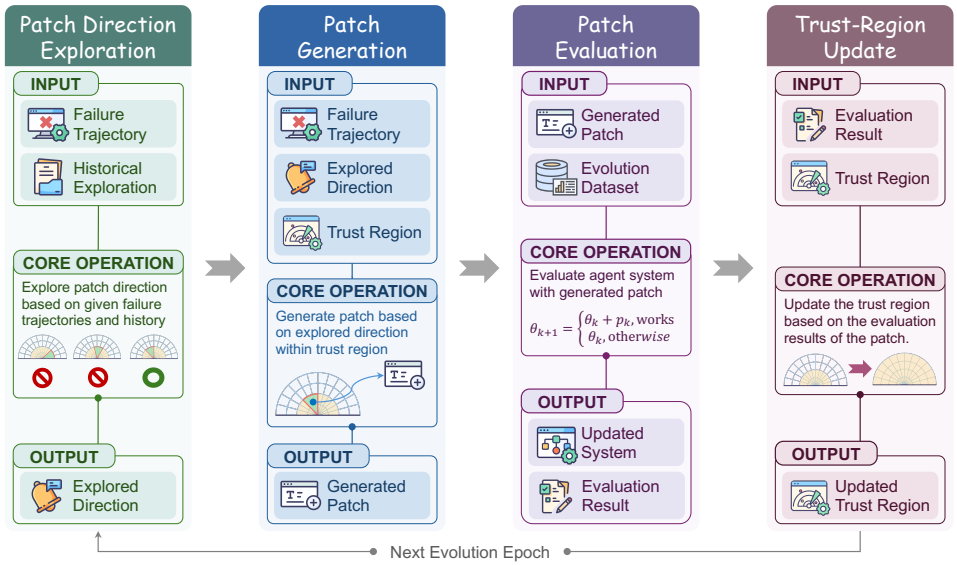


Figure 3: The illustration of TRUSTEVO, where each epoch consists of four steps: *(i) Patch Direction Exploration* explores the patch generation direction with failure trajectories. *(ii) Patch Generation* generates the patch based on explored direction within trust region. *(iii) Patch Evaluation* evaluates the generated patch and updates the agent system correspondingly. *(iv) Trust-Region Update* updates the trust region based on the evaluation results.

5 INSPIRATION: AGENTIC EVOLUTION WITH TRUST REGION

5.1 TRUSTEVO

The analysis in §3 shows that restricting the size of each patch and the number of affected components can improve evolution effectiveness and efficiency, as well as the identifiability of patch directions. Meanwhile, prior work suggests that identifying effective improvement directions is a major source of cost in agentic evolution (Shang et al., 2025; Li et al., 2025). Motivated by these observations, we propose TRUSTEVO, which dynamically controls the patch size and the number of affected components to jointly improve effectiveness, efficiency, and identifiability. Following trust-region optimization (Schulman et al., 2015), TRUSTEVO restricts each candidate patch to the current trust region and adapts the region according to the evaluation of the patch. An illustration of TRUSTEVO is shown in Figure 3, and the prompts used by TRUSTEVO are provided in Appendix E.

5.1.1 PATCH DIRECTION EXPLORATION

This step is designed to determine the overall direction of improvement for the current epoch based on the failure trajectories produced by the current agent system and their corresponding evaluation results. Besides, TRUSTEVO also maintains the improvement directions explored in previous epochs and their evaluation results. Directions that previously cause performance regressions or excessive evolution costs are constrained, whereas directions that repeatedly produce reliable gains are prioritized. Based on the above input, TRUSTEVO identifies the components responsible for the observed failures and summarizes their component-level problems. The resulting failure attribution and improvement direction guide the subsequent generation of a concrete patch.

5.1.2 PATCH GENERATION WITH TRUST REGION

Given the explored improvement direction and the corresponding failure trajectories, TRUSTEVO generates a candidate patch p_k. Following trust-region optimization, the candidate must remain within the current trust region. Specifically, the trust region constrains both the patch size, measured by its token count, and the number of components whose writable parameters may be modified. A larger trust region permits a higher token budget and allows the patch to affect more components, whereas a smaller trust region enforces a more localized update. Because current LLMs cannot precisely control the length of their outputs, we implement the patch-size constraint primarily by specifying an explicit maximum token budget in the patch-generation instruction. The instruction also specifies the allowed number or set of components that may be modified.

5.1.3 PATCH EVALUATION

After obtaining the candidate patch p_k, TRUSTEVO applies it to the current agent system θ_k and evaluates the resulting candidate system $\theta_k + p_k$ under the fixed evaluation protocol. The patch is accepted only when it produces a reliable performance improvement without unacceptable regressions. Accordingly, the agent state is updated as

$$\theta_{k+1} = \begin{cases} \theta_k + p_k, & \text{if } p_k \text{ is accepted,} \\ \theta_k, & \text{otherwise.} \end{cases} \tag{27}$$

Thus, an ineffective patch is discarded, and the next epoch continues from the original agent system.

5.1.4 TRUST-REGION UPDATE

In addition to updating the agent system, TRUSTEVO adapts the trust region according to the patch-evaluation result. If the patch is accepted, the trust region is enlarged to permit a moderately larger patch or a broader component-edit scope in the next epoch. Because this expansion follows a measured improvement and remains centered on an empirically validated direction, it can accelerate evolution without immediately reverting to unrestricted editing. If the patch is rejected, the trust region is reduced, forcing the patch generator to produce smaller and more localized updates. The trust-region update depends on the magnitude of the performance change, where the new trust region is adjusted by a factor equal to the ratio of the new performance to the previous performance. This conservative adjustment helps control evolution error and cost while making subsequent failure attribution less ambiguous. The updated trust region is then used for the next epoch $k + 1$.

5.2 EXPERIMENT

To validate the effectiveness of TRUSTEVO, we conduct agentic evolution experiments under the settings described in §4.1. To ensure robustness, for each setting, we use three-fold cross-validation, where we conduct three runs, and each run uses one-third of the data held out as the test set while using the remaining two-thirds for evaluation. The experimental results are presented in Table 2, which shows that TRUSTEVO achieves the highest pass rate and the lowest evolution cost compared with all baselines in all settings. On average, it reaches a pass rate of 37.0%, outperforming the base systems and ACE by 17.9 and 3.9 percentage points, respectively, while using 16.5% fewer tokens than ACE. This confirms that restricting patch magnitude and affected components, together with an adaptive trust region, improves both evolution effectiveness and efficiency, thereby demonstrating the effectiveness of the guidance provided by our framework.

Table 2: Instance-level pass rates and overall token used on τ^2-Bench, Terminal-Bench-Hard (TB-H), and WebArena-Verified-Hard (WA-VH). *#Tok.* denotes the average tokens used for evolution. Base denotes the baseline agentic system of each benchmark. We introduce all baselines in Appendix F. All evolution methods are averaged on three-fold runs. The best result under each setting is marked in **bold**. We also adapt the ablation study of TRUSTEVO in Appendix I.

Model	Method	τ^2		TB-H		WA-VH	
		Pass (%) ↑	#Tok. (10^8) ↓	Pass (%) ↑	#Tok. (10^8) ↓	Pass (%) ↑	#Tok. (10^8) ↓
GPT-5.4-NANO	Base	26.5	–	16.9	–	10.9	–
	GEPA	38.0 ± 1.6	0.91	25.9 ± 2.2	0.84	16.3 ± 1.2	0.97
	ACE	44.1 ± 2.2	0.62	32.6 ± 2.3	0.59	20.2 ± 1.8	0.68
	ADAS	39.8 ± 2.2	1.18	30.3 ± 3.4	1.27	15.9 ± 1.3	1.31
	TRUSTEVO	$\mathbf{46.2 \pm 2.2}$	**0.51**	$\mathbf{37.1 \pm 3.4}$	**0.49**	$\mathbf{24.8 \pm 1.8}$	**0.56**
CLAUDE-HAIKU-4.5	Base	25.1	–	14.6	–	14.3	–
	GEPA	35.5 ± 1.9	0.98	23.6 ± 3.0	0.92	20.5 ± 1.3	1.05
	ACE	41.2 ± 2.2	0.67	29.2 ± 2.3	0.63	25.6 ± 2.0	0.74
	ADAS	37.3 ± 2.5	1.26	27.0 ± 3.4	1.38	19.4 ± 1.3	1.42
	TRUSTEVO	$\mathbf{44.4 \pm 2.2}$	**0.55**	$\mathbf{34.8 \pm 3.3}$	**0.53**	$\mathbf{29.5 \pm 1.8}$	**0.61**
DEEPSEEK-V4-FLASH	Base	29.0	–	21.3	–	12.8	–
	GEPA	40.1 ± 1.6	1.06	28.1 ± 2.5	1.01	18.6 ± 1.2	1.14
	ACE	45.9 ± 2.2	0.73	35.9 ± 3.6	0.70	23.3 ± 2.0	0.81
	ADAS	42.3 ± 2.2	1.35	33.7 ± 3.4	1.49	17.8 ± 1.8	1.54
	TRUSTEVO	$\mathbf{48.4 \pm 2.2}$	**0.60**	$\mathbf{40.5 \pm 4.1}$	**0.64**	$\mathbf{27.1 \pm 2.4}$	**0.66**

6 RELATED WORKS

Agent evolution describes the shift from fixed, human-engineered agents to adaptive systems that improve policies, memory, tools, and architectures through interaction and automated search (ang Gao et al., 2026). Its roots lie in evolutionary robotics and neuroevolution (Sims, 1994; Stanley & Miikkulainen, 2002), followed by self-play, population-based reinforcement learning, and open-ended co-evolution, which showed that curricula and capabilities can emerge jointly (Silver et al., 2018; Jaderberg et al., 2019; Wang et al., 2019). With LLM agents, evolution shifted toward language-mediated cognition: ReAct and Toolformer enabled reasoning–action loops and tool use (Yao et al., 2023; Schick et al., 2023), while later work introduced memory, multi-agent collaboration, reflection, experience abstraction, and reusable skills (Park et al., 2023; Li et al., 2023; Wu et al., 2024; Shinn et al., 2023; Madaan et al., 2023; Zhao et al., 2024; Wang et al., 2024). Recent methods evolve prompts, populations, interaction graphs, workflows, and agent architectures (Fernando et al., 2024; Xi et al., 2025; Yuan et al., 2025; Zhuge et al., 2024; Hu et al., 2025; Li et al., 2024; Zhang et al., 2025a; Shang et al., 2025). The latest stage targets open-ended self-improvement through self-modifying code, autonomous task generation, co-evolved curricula and executors, and increasingly autonomous adaptation mechanisms (Zhang et al., 2026; Xia et al., 2026; Yang et al., 2026).

However, the above works primarily focus on performance while lacking an analysis of how different factors influence the evolution process, which limits further advancements in this area. Therefore, in this paper, we propose a control-theoretic framework for analyzing agentic evolution. Based on this framework, we investigate how different factors affect the effectiveness, efficiency, and identifiability of the evolution process, thereby providing guidance for the design of future works.

7 CONCLUSION

We present a control-theoretic framework for understanding agent evolution as a discrete-time closed-loop process. The framework jointly analyzes evolution effectiveness, efficiency, and identifiability, revealing that effective evolution depends not only on final performance but also on how patches are generated and validated. These findings motivated TRUSTEVO, which combines failure-driven direction exploration, constrained patch generation, explicit acceptance tests, and adaptive trust-region updates. Experiments across three benchmarks show that TRUSTEVO consistently improves both pass rate and token efficiency, while ablations confirm the complementary roles of direction exploration and trust-region control. Overall, the results establish our framework as a useful foundation for analyzing and designing more effective and interpretable evolution systems.

AI USE STATEMENT

We primarily use AI tools for paper polishing and coding assistance. All research ideas presented in this paper are independently conceived by the authors, and the core manuscript is written by the authors. All uses of AI tools in this work comply with the ICLR 2027 policy.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Huan ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru WANG, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=CTr3bovS5F. Survey Certification.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik R Narasimhan. $\tauˆ2$-bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OC2z7iSQKa.

Ali Eslami and Jiangbo Yu. A control-theoretic foundation for agentic systems, 2026. URL https://arxiv.org/abs/2603.10779.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019. doi: 10.1126/science.aau6249. URL https://www.science.org/doi/abs/10.1126/science.aau6249.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: communicative agents for "mind" exploration of large language model society. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Yu Li, Lehui Li, Zhihao Wu, Qingmin Liao, Jianye Hao, Kun Shao, Fengli Xu, and Yong Li. Agentswift: Efficient llm agent design via value-guided hierarchical search, 2025. URL https://arxiv.org/abs/2506.06017.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. Autoflow: Automated workflow generation for large language model agents. *CoRR*, abs/2407.12821, 2024. URL https://doi.org/10.48550/arXiv.2407.12821.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Jenia Jitsev, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Komeil Nosrati, Aleksei Tepljakov, Juri Belikov, and Eduard Petlenkov. When control meets large language models: From words to dynamics. *Engineering Applications of Artificial Intelligence*, 178:115119, August 2026. ISSN 0952-1976. doi: 10.1016/j.engappai.2026.115119. URL http://dx.doi.org/10.1016/j.engappai.2026.115119.

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*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Kerri Prinos, Lilianne Brush, Cameron Denton, Zhanqi Wang, Joshua Knox, Snehal Antani, Anton Foltz, and Amy Villaseñor. Stable agentic control: Tool-mediated llm architecture for autonomous cyber defense, 2026. URL https://arxiv.org/abs/2605.03034.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schulman15.html.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=mPdmDYIQ7f.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap,

Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/abs/10.1126/science.aar6404.

Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pp. 15–22, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916670. doi: 10.1145/192161.192167. URL https://doi.org/10.1145/192161.192167.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002. ISSN 1063-6560. doi: 10.1162/106365602320169811. URL https://doi.org/10.1162/106365602320169811.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, 2019. URL https://arxiv.org/abs/1901.01753.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evaluating and training large language model-based agents across diverse environments. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27914–27961, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1355. URL https://aclanthology.org/2025.acl-long.1355/.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. In *ICLR 2026 Workshop on Lifelong Agents: Learning, Aligning, Evolving*, 2026. URL https://openreview.net/forum?id=OWz5JiJw5M.

Yongjin Yang, Sinjae Kang, Juyong Lee, Dongjun Lee, Se-Young Yun, and Kimin Lee. Toward self-evolving systems of LLM agents through exploration and iterative feedback, 2026. URL https://openreview.net/forum?id=uO3gGxzu8k.

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*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.315. URL https://aclanthology.org/2025.naacl-long.315/.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=pUpzQZTvGY.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=z5uVAKwmjf.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2025b. URL https://arxiv.org/abs/2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN 978-1-57735-887-9. doi: 10.1609/aaai.v38i17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uTC9AFXIhg.

A PROOFS OF THEOREMS IN §3

In this section, we present proofs of theorems in §3. Throughout this section, all vector norms are Euclidean norms, and all matrix norms are their induced operator norms, consistent with Section 3.

Proof 1 (Proof of Theorem 1) *Because the evolution update is $\theta_{k+1} = \theta_k + p_k$, we have*

$$e_{k+1} = e(\theta_k + p_k). \tag{28}$$

By assumption, the line segment between θ_k and $\theta_k + p_k$ lies in $\mathcal{K}$. Applying the fundamental theorem of calculus along this segment gives

$$e(\theta_k + p_k) - e(\theta_k) = \int_0^1 J(\theta_k + tp_k)p_k \, dt. \tag{29}$$

Hence

$$\begin{aligned} &\big\|e(\theta_k + p_k) - e_k - J(\theta_k)p_k\big\| \\ &\quad\le \int_0^1 \big\|J(\theta_k + tp_k) - J(\theta_k)\big\|\|p_k\| \, dt \\ &\quad\le \int_0^1 Lt\|p_k\|^2 \, dt = \frac{L}{2}\|p_k\|^2, \end{aligned} \tag{30}$$

where the second inequality uses the L-Lipschitz continuity of J on $\mathcal{K}$. Combining Equations 28 and 30 with the assumed linearized contraction and patch-radius bound yields

$$\begin{aligned} \|e_{k+1}\| &\le \|e_k + J(\theta_k)p_k\| + \frac{L}{2}\|p_k\|^2 \\ &\le \eta\|e_k\| + \frac{L}{2}\overline{\Delta}^2. \end{aligned} \tag{31}$$

Iterating this recursion gives, for every $k \ge 1$,

$$\begin{aligned} \|e_k\| &\le \eta^k\|e_0\| + \frac{L\overline{\Delta}^2}{2}\sum_{j=0}^{k-1}\eta^j \\ &= \eta^k\|e_0\| + \frac{L\overline{\Delta}^2}{2(1-\eta)}\left(1-\eta^k\right). \end{aligned} \tag{32}$$

Since $0 \le \eta < 1$, taking the limit superior as $k \to \infty$ yields

$$\limsup_{k\to\infty}\|e_k\| \le \frac{L\overline{\Delta}^2}{2(1-\eta)},$$

proving the theorem.

Proof 2 (Proof of Theorem 2) *Repeated application of the contraction condition gives*

$$\|e_k\| \le q^k\|e_0\|, \qquad k \ge 0. \tag{33}$$

Using the patch-to-error bound at the same epoch,

$$\|p_k\|^2 \le c\|e_k\|^2 \le cq^{2k}\|e_0\|^2. \tag{34}$$

Therefore, for any finite horizon $K > 0$,

$$\begin{aligned} \mathcal{E}_K &= \sum_{k=0}^{K-1}\left(\|e_k\|^2 + \|p_k\|^2\right) \\ &\le (1+c)\|e_0\|^2\sum_{k=0}^{K-1}q^{2k} \\ &= \frac{(1+c)(1-q^{2K})}{1-q^2}\|e_0\|^2. \end{aligned} \tag{35}$$

Because all summands are nonnegative, $\mathcal{E}_K$ is nondecreasing in K. Since $0 \le q < 1$, letting $K \to \infty$ in Equation 35 gives

$$\mathcal{E}_\infty \le \frac{1+c}{1-q^2}\|e_0\|^2,$$

as claimed.

Proof 3 (Proof of Theorem 3) *Let*

$$D_g = \operatorname{diag}(\|g_1\|, \dots, \|g_m\|)$$

and define the normalized Gram matrix

$$R = D_g^{-1} W_{\text{patch}} D_g^{-1}.$$

Because $\|g_i\| \ge \alpha > 0$, D_g is nonsingular. Moreover,

$$R_{ii} = 1, \qquad |R_{ij}| = \frac{|g_i^\top g_j|}{\|g_i\|\,\|g_j\|} \le \mu \quad (i \ne j).$$

The Gershgorin circle theorem therefore implies

$$\lambda_{\min}(R) \ge 1 - (m-1)\mu. \tag{36}$$

The assumed condition $(m-1)\mu < 1$ makes the right-hand side positive. For any $x \in \mathbb{R}^m$,

$$\begin{aligned} x^\top W_{\text{patch}} x &= (D_g x)^\top R (D_g x) \\ &\ge \big(1-(m-1)\mu\big)\|D_g x\|^2 \\ &\ge \alpha^2\big(1-(m-1)\mu\big)\|x\|^2. \end{aligned} \tag{37}$$

Taking the minimum Rayleigh quotient over all nonzero x yields

$$\lambda_{\min}(W_{\text{patch}}) \ge \alpha^2\big(1-(m-1)\mu\big),$$

proving the theorem. The case $m = 1$ is included by the convention $\mu = 0$.

B Additional System Properties

The main analysis in Section 3 focuses on effectiveness, efficiency, and identifiability because these properties directly guide the design of agentic evolution. To better demonstrate the effectiveness of our framework, we record three complementary closed-loop properties below. They are not required by the main theorems. In particular, the state observability considered here is distinct from the patch identifiability in Section 3.3, where observability asks whether the writable state can be recovered from an output sequence, whereas identifiability asks whether different candidate patch directions can be distinguished by their local output effects.

For a fixed hyperparameter $u = (\psi, \phi, D)$, define

$$p_u(\theta) = P_\psi\left(\begin{bmatrix} \theta \\ \phi \\ D \\ \mathcal{Y}(D;\phi,\theta) \\ \Gamma(D) \end{bmatrix}\right), \qquad F_u(\theta) = \theta + p_u(\theta). \tag{38}$$

Thus, an evolution run with fixed u follows $\theta_{k+1} = F_u(\theta_k)$. When u is treated as a differentiation variable below, we assume that the admissible family of hyperparameters is represented locally by finite-dimensional Euclidean coordinates. Varying u then describes nearby evolution runs. Within any single run, u remains fixed as required by Section 2.

B.1 Stability

We first give a sufficient condition for the convergence of the writable-parameter trajectory under a fixed hyperparameter u.

Theorem 4 (Stability of agent evolution) *Assume that $p_u : \mathbb{R}^n \to \mathbb{R}^n$ is continuous. Suppose there exists $\theta^\star \in \mathbb{R}^n$ such that*

$$2\langle \theta - \theta^\star, p_u(\theta)\rangle + \|p_u(\theta)\|^2 < 0, \qquad \forall \theta \neq \theta^\star. \tag{39}$$

Then, for every initial parameter $\theta_0 \in \mathbb{R}^n$, the trajectory generated by $\theta_{k+1} = F_u(\theta_k)$ converges to $\theta^\star$.

Proof 4 *Define the Lyapunov candidate*

$$V(\theta) = \|\theta - \theta^\star\|^2,$$

and let

$$\Delta V(\theta) := V(F_u(\theta)) - V(\theta).$$

Since $F_u(\theta) = \theta + p_u(\theta)$, direct expansion gives

$$\begin{aligned} \Delta V(\theta) &= \|\theta - \theta^\star + p_u(\theta)\|^2 - \|\theta - \theta^\star\|^2 \\ &= 2\langle \theta - \theta^\star, p_u(\theta)\rangle + \|p_u(\theta)\|^2. \end{aligned} \tag{40}$$

Hence, by Equation 39,

$$\Delta V(\theta) < 0, \qquad \forall \theta \neq \theta^\star. \tag{41}$$

We first show that $\theta^\star$ is a fixed point of F_u. Since p_u is continuous, ΔV is continuous. For any sequence $\{\theta^{(j)}\}_{j\geq 1}$ satisfying $\theta^{(j)} \neq \theta^\star$ and $\theta^{(j)} \to \theta^\star$, Equation 41 yields $\Delta V(\theta^{(j)}) < 0$ for every j. By continuity,

$$\Delta V(\theta^\star) = \lim_{j\to\infty} \Delta V(\theta^{(j)}) \leq 0.$$

On the other hand, substituting $\theta = \theta^\star$ into Equation 40 gives

$$\Delta V(\theta^\star) = \|p_u(\theta^\star)\|^2 \geq 0.$$

Therefore

$$\Delta V(\theta^\star) = 0, \qquad p_u(\theta^\star) = 0,$$

and consequently $F_u(\theta^\star) = \theta^\star$.

Now fix an arbitrary initial state $\theta_0 \in \mathbb{R}^n$ and let

$$V_k := V(\theta_k).$$

If $\theta_k = \theta^\star$ for some k, then the fixed-point property implies $\theta_j = \theta^\star$ for all $j \geq k$, and the desired convergence follows. It therefore remains to consider the case in which $\theta_k \neq \theta^\star$ for every k. From Equations 40–41,

$$V_{k+1} - V_k = \Delta V(\theta_k) < 0, \qquad \forall k \geq 0.$$

Thus $\{V_k\}$ is strictly decreasing and bounded below by zero. Hence there exists $V_\infty \geq 0$ such that

$$V_k \longrightarrow V_\infty.$$

Suppose, for contradiction, that $V_\infty > 0$. Since $\{V_k\}$ is decreasing,

$$V_\infty \leq V_k \leq V_0, \qquad \forall k \geq 0.$$

Therefore every θ_k belongs to

$$\mathcal{A} := \left\{\theta \in \mathbb{R}^n : V_\infty \leq \|\theta - \theta^\star\|^2 \leq V_0\right\}.$$

The set $\mathcal{A}$ is closed and bounded, and hence compact. Moreover, $V_\infty > 0$ implies $\theta^\star \notin \mathcal{A}$. By Equation 41, $\Delta V(\theta) < 0$ for every $\theta \in \mathcal{A}$. Since ΔV is continuous and $\mathcal{A}$ is compact, it attains a maximum on $\mathcal{A}$. This maximum is strictly negative; hence there exists $\delta > 0$ such that

$$\Delta V(\theta) \leq -\delta, \qquad \forall \theta \in \mathcal{A}.$$

In particular,

$$V_{k+1} \leq V_k - \delta, \qquad \forall k \geq 0.$$

Iterating this inequality gives

$$V_k \leq V_0 - k\delta,$$

which is negative for sufficiently large k, contradicting $V_k \geq 0$. Therefore $V_\infty = 0$.

Consequently,

$$\|\theta_k - \theta^\star\|^2 = V_k \longrightarrow 0,$$

and hence

$$\theta_k \longrightarrow \theta^\star.$$

Since θ_0 was arbitrary, the result holds for every initial state.

B.2 Endpoint Reachability Across Evolution Runs

Since the hyperparameter u is fixed within each evolution run, it is not a time-varying control input. A compatible notion of reachability is to ask which nearby endpoints can be obtained by making a nearby fixed hyperparameter choice before a run begins. For a fixed initial state θ_0 and horizon $H \geq 1$, define

$$\theta_{k+1} = F_u(\theta_k), \qquad \Phi_H(\theta_0; u) = \theta_H. \tag{42}$$

Theorem 5 (Local endpoint reachability from a full-rank hyperparameter map) *Let the admissible hyperparameters be represented locally by an open set $\mathcal{U} \subseteq \mathbb{R}^m$, with $m \geq n$, and suppose that $u \mapsto \Phi_H(\theta_0; u)$ is C^1 near $u^\circ \in \mathcal{U}$. If*

$$\operatorname{rank} \frac{\partial \Phi_H(\theta_0; u)}{\partial u}\bigg|_{u=u^\circ} = n, \tag{43}$$

then there exist neighborhoods $\mathcal{U}_0$ of u° and $\mathcal{V}_0$ of $\theta_H^\circ = \Phi_H(\theta_0; u^\circ)$ such that, for every $\bar{\theta} \in \mathcal{V}_0$, there exists a fixed hyperparameter $u \in \mathcal{U}_0$ satisfying $\Phi_H(\theta_0; u) = \bar{\theta}$.

Proof 5 *Define the endpoint map*

$$E : \mathcal{U} \to \mathbb{R}^n, \qquad E(u) := \Phi_H(\theta_0; u).$$

By assumption, E is C^1 in a neighborhood of u° and

$$\operatorname{rank} DE(u^\circ) = n.$$

Since $DE(u^\circ) \in \mathbb{R}^{n \times m}$ has full row rank and $m \geq n$, it contains an invertible $n \times n$ minor. After a permutation of the local coordinates of u, write

$$u = (v, w), \qquad v \in \mathbb{R}^n, \qquad w \in \mathbb{R}^{m-n},$$

and correspondingly $u^\circ = (v^\circ, w^\circ)$, such that

$$D_v E(v^\circ, w^\circ) = \frac{\partial \Phi_H(\theta_0; (v, w))}{\partial v}\bigg|_{(v,w)=(v^\circ, w^\circ)} \tag{44}$$

is nonsingular.

Because $\mathcal{U}$ is open and $(v^\circ, w^\circ) \in \mathcal{U}$, there exist open neighborhoods $\mathcal{V} \subseteq \mathbb{R}^n$ of v° and $\mathcal{W} \subseteq \mathbb{R}^{m-n}$ of w° such that

$$\mathcal{V} \times \mathcal{W} \subseteq \mathcal{U}.$$

Define the restricted map

$$G : \mathcal{V} \to \mathbb{R}^n, \qquad G(v) := E(v, w^\circ) = \Phi_H(\theta_0; (v, w^\circ)).$$

The map G is C^1, and by Equation 44,

$$DG(v^\circ) = D_v E(v^\circ, w^\circ)$$

is nonsingular.

The inverse function theorem therefore guarantees the existence of open neighborhoods $\mathcal{V}_0 \subseteq \mathcal{V}$ of v° and $\mathcal{O}_0 \subseteq \mathbb{R}^n$ of

$$G(v^\circ) = E(u^\circ) = \Phi_H(\theta_0; u^\circ) = \theta_H^\circ$$

such that

$$G|_{\mathcal{V}_0} : \mathcal{V}_0 \to \mathcal{O}_0$$

is a C^1 diffeomorphism.

Choose any open neighborhood $\mathcal{W}_0 \subseteq \mathcal{W}$ of w° and define

$$\mathcal{U}_0 := \mathcal{V}_0 \times \mathcal{W}_0, \qquad \mathcal{V}_0^\theta := \mathcal{O}_0.$$

Then $\mathcal{U}_0$ is a neighborhood of u° contained in $\mathcal{U}$, and $\mathcal{V}_0^\theta$ is a neighborhood of θ_H°.

Let an arbitrary target $\bar{\theta} \in \mathcal{V}_0^\theta$ be given. Since $G|_{\mathcal{V}_0}$ is bijective onto $\mathcal{V}_0^\theta$, there exists a unique

$$\bar{v} = (G|_{\mathcal{V}_0})^{-1}(\bar{\theta}) \in \mathcal{V}_0.$$

Set

$$\bar{u} := (\bar{v}, w^\circ).$$

Because $w^\circ \in \mathcal{W}_0$, $\bar{u} \in \mathcal{U}_0$, and

$$\Phi_H(\theta_0; \bar{u}) = E(\bar{v}, w^\circ) = G(\bar{v}) = \bar{\theta}.$$

Thus every endpoint in a neighborhood of θ_H° can be reached by a sufficiently nearby fixed hyperparameter choice.

B.3 State Observability

Finally, fix a known hyperparameter $u = (\psi, \phi, D)$ and consider whether the initial writable parameters can be recovered from the outputs generated during the resulting evolution trajectory. For an initial state θ, define recursively

$$\theta^{(0)}(\theta) = \theta, \qquad \theta^{(k+1)}(\theta) = F_u(\theta^{(k)}(\theta)),$$

and define the finite-horizon output map

$$\mathcal{H}_{H,u}(\theta) = \begin{bmatrix} \mathcal{Y}(D; \phi, \theta^{(0)}(\theta)) \\ \vdots \\ \mathcal{Y}(D; \phi, \theta^{(H)}(\theta)) \end{bmatrix}. \tag{45}$$

Theorem 6 (Local finite-step state observability from a full-rank output map) *Let $H \geq 0$ and suppose that $\mathcal{H}_{H,u}$ is C^1 near an initial state $\theta_0 \in \mathbb{R}^n$. If*

$$\operatorname{rank} \frac{\partial \mathcal{H}_{H,u}(\theta)}{\partial \theta}\bigg|_{\theta=\theta_0} = n, \tag{46}$$

then there exists a neighborhood $\mathcal{N}$ of θ_0 on which the known output sequence $y_0, \ldots, y_H$ uniquely determines the initial writable parameters. Thus, the closed-loop evolution process is locally finite-step state observable at θ_0 for the fixed known hyperparameter u.

Proof 6 *Let the stacked output map take values in $\mathbb{R}^q$ for some finite $q \geq n$, and write*

$$H(\theta) := \mathcal{H}_{H,u}(\theta).$$

By assumption, H is C^1 in a neighborhood of θ_0 and

$$\operatorname{rank} DH(\theta_0) = n.$$

Hence the $q \times n$ matrix $DH(\theta_0)$ has full column rank. Therefore there exists a set of n row indices

$$I = \{i_1, \ldots, i_n\} \subseteq \{1, \ldots, q\}$$

such that the corresponding $n \times n$ row submatrix of $DH(\theta_0)$ is nonsingular.

Let

$$\Pi_I : \mathbb{R}^q \to \mathbb{R}^n$$

denote the coordinate projection onto the coordinates indexed by I, and define

$$G := \Pi_I \circ H.$$

Since Π_I is linear and H is C^1, G is C^1. Moreover,

$$DG(\theta_0) = \Pi_I DH(\theta_0),$$

which is precisely the selected nonsingular $n \times n$ minor. Thus

$$\det DG(\theta_0) \neq 0.$$

By the inverse function theorem, there exist open neighborhoods $\mathcal{N}$ of θ_0 and $\mathcal{O}$ of $G(\theta_0)$ such that

$$G|_{\mathcal{N}} : \mathcal{N} \to \mathcal{O}$$

is a C^1 diffeomorphism. In particular, $G|_{\mathcal{N}}$ is injective.

We now prove that $H|_{\mathcal{N}}$ is also injective. Let $\theta, \tilde{\theta} \in \mathcal{N}$ satisfy

$$H(\theta) = H(\tilde{\theta}).$$

Applying the projection Π_I to both sides yields

$$G(\theta) = \Pi_I H(\theta) = \Pi_I H(\tilde{\theta}) = G(\tilde{\theta}).$$

Since $G|_{\mathcal{N}}$ is injective,

$$\theta = \tilde{\theta}.$$

Therefore

$$H|_{\mathcal{N}}$$

is one-to-one.

For the fixed known hyperparameter $u = (\psi, \phi, D)$, an evolution run initialized at θ produces the stacked output sequence

$$\begin{bmatrix} y_0 \\ \vdots \\ y_H \end{bmatrix} = \mathcal{H}_{H,u}(\theta) = H(\theta).$$

Since H is injective on $\mathcal{N}$, no two distinct initial writable states in $\mathcal{N}$ can generate the same finite output sequence. Hence the observed sequence $y_0, \ldots, y_H$ uniquely determines the initial writable parameters within $\mathcal{N}$.

Therefore the closed-loop evolution process is locally finite-step state observable at θ_0 for the fixed known hyperparameter u.

C Instantiations of Mainstream Agent Systems

We illustrate how our formalization captures three representative agent architectures: ReAct, Generative Agents, and AutoGen. These systems respectively represent tool-augmented single-agent reasoning, memory-augmented decision making, and multi-agent collaboration.

ReAct. ReAct (Yao et al., 2023) agents interleave language-model reasoning with actions in an external environment. In our formalization, the components consist of a reasoning component f_{reason} and one component f_i for each available tool or environment interface. Given the trajectory h_t, the planner $C(h_t; \theta_k^D)$ determines whether the agent should continue reasoning, invoke a particular tool, or return the final answer. This decision is represented by a one-hot α_t. The reasoning component produces the next thought, action proposal, or final response, whereas a tool component executes its corresponding action and returns the resulting observation as z_t. The fixed channel ϕ_i specifies the capability and interface of each tool. The writable parameters θ_k^D may encode the routing instructions, while $\theta_{i,k}$ may include component prompts, tool descriptions, demonstrations, or editable wrapper code. The final ReAct response is therefore represented by $A(\xi; \phi, \theta_k) = z_{\tau_{\max}}$.

Generative Agents. Generative Agents (Park et al., 2023) combine a memory stream with retrieval, reflection, and planning mechanisms. We represent these mechanisms as separate components, such as a memory-retrieval component, a reflection component, and a planning or action-generation component. Based on the current observation and previous intermediate outputs in h_t, the planner selects the operation required at step t. The retrieval component returns relevant past experiences, the reflection component synthesizes higher-level conclusions, and the planning component generates a plan or the next environmental action. Newly observed events, retrieved memories, and generated reflections are included in z_t and consequently become part of the subsequent history. The fixed channels ϕ_i distinguish the semantic roles of retrieval, reflection, and planning, while their writable parameters may contain retrieval rules, prompts, scoring functions, or persistent memory records. Thus, the complete memory–reflection–planning loop is represented as a finite sequence of component selections and outputs.

AutoGen. AutoGen (Wu et al., 2024) constructs agent applications through conversations among multiple customizable agents. In our formalization, each conversable agent is treated as one component f_i, and the session history h_t corresponds to the shared conversation transcript, including previous messages, tool results, and human feedback. The planner C represents the conversation manager or speaker-selection policy. At each step, it assigns a one-hot weight to the next agent allowed to speak; agents that are not eligible under the conversation protocol receive zero weight. The selected component generates the next message or action from the shared history and its local configuration. The fixed channel ϕ_i specifies the agent's role and permitted interfaces, whereas $\theta_{i,k}$ may include its system prompt, demonstrations, local memory, tools, or executable code. The planner

Table 3: The calculation of properties and factors.

Property	Symbol	Calculation
Base	p_k	The patch of epoch k, where $\|p_k\|$ is calculated as the token number of p_k.
	e_k	The error of epoch k, where $\|e_k\|$ denotes the value of the error.
Effectiveness	$\bar{\Delta}$	Compute the patch magnitude at each evolution step as $\Delta_k = \|p_k\|$, and average it over the trajectory: $\bar{\Delta} = \frac{1}{K}\sum_{k=0}^{K-1}\Delta_k$.
	η	Compute the residual ratio $\eta_k = \|e_{k+1}\|/\|e_k\|$ at each step, and estimate the trajectory-level contraction rate using the mean of these ratios.
	e_∞	Evaluate the evolved system on the same validation set after the final evolution step.
Efficiency	e_0	Evaluate the system on the validation set before evolution and compute the norm of the initial residual vector $\|e_0\|$.
	q	Record $\|e_k\|$ after each evolution epoch and estimate its geometric decay rate, e.g., by fitting $\log\|e_k\|$ against the epoch index.
	c	Compute $c_k = \|p_k\|^2/\|e_k\|^2$ at each evolution step and use the median across the trajectory as the empirical patch-to-error ratio.
	$\widehat{\mathcal{E}}$	Accumulate the residual and patch costs over the full trajectory: $\widehat{\mathcal{E}} = \sum_{k=0}^{K-1}\left(\|e_k\|^2 + \|p_k\|^2\right)$.
Identifiability	α	Apply the same small normalized perturbation to each candidate components, measure its response vector g_i on a fixed feedback set, and take the smallest response norm: $\alpha = \min_i \|g_i\|$.
	μ	Compute the absolute cosine similarity between every pair of measured responses g_i and g_j, and take the maximum pairwise similarity.
	m	Count the number of components that remain simultaneously plausible for the evaluated failure under the fixed attribution rule.
	$\lambda_{\min}$	Stack the measured response vectors as $G = [g_1, \ldots, g_m]$ and compute the smallest eigenvalue of $G^\top G$, i.e., $\lambda_{\min} = \lambda_{\min}(G^\top G)$.

parameters θ_k^D encode speaker-transition and termination rules. Human-proxy and tool-execution agents can be included as additional components, and the final answer produced by the conversation is represented by $z_{\tau_{\max}}$.

D Calculation of Properties and Factors in §3

We summarize the calculation of properties and factors in Table 3.

E Prompt of TrustEvo

The prompts used by TrustEvo are shown in Table 4 and Table 5.

F Baseline Methods

GEPA. GEPA (Genetic-Pareto) (Agrawal et al., 2026) is a reflective prompt-optimization method that evolves textual parameters using execution feedback. It collects agent trajectories and evaluation results, then uses an LLM-based reflector to diagnose failures and propose targeted prompt mutations. A genetic Pareto search evaluates the resulting candidates and maintains a diverse set of prompts that perform well on different subsets of examples.

ACE. ACE (Agentic Context Engineering) (Zhang et al., 2025b) treats an LLM system's context as an evolving structured playbook. Its framework consists of a *Generator*, which produces task

Table 4: The prompt of patch direction exploration of TRUSTEVO.

Patch Direction Exploration

You are an expert in diagnosing and improving agent systems. Your task is to identify one focused and actionable patch direction from a collection of failed trajectories.
Input
Failed-trajectory directory: `<FAILED_TRAJECTORIES_DIR>`
Recursively inspect the files in the given directory. A trajectory may contain the task specification, agent actions, intermediate reasoning or state, tool calls, environment observations, the final output, and evaluator feedback.
Analyze the failed trajectories according to the following requirements:

1. Identify failure patterns that recur across multiple trajectories, rather than overfitting to an isolated example.
2. Distinguish the underlying cause of a failure from its downstream symptoms.
3. Attribute the failures to the smallest plausible set of writable agent components, such as the planner, retrieval module, memory module, skill module, tool-use policy, or response-generation policy.
4. Prioritize failures that are frequent, severe, and addressable through a localized modification.
5. Propose exactly one patch direction. The direction should describe what behavioral or component-level change should be made, but should not contain a concrete patch or implementation.
6. Avoid broad instructions such as "improve reasoning," "be more robust," or "handle errors better." The proposed direction must be specific enough to guide patch generation.

Return exactly one concise natural-language paragraph using the following structure:
Target component(s): identify the component or components most likely responsible. **Failure pattern:** summarize the recurring observable failures. **Root cause:** explain the most likely component-level cause and the supporting evidence. **Patch direction:** describe one specific and localized change that should address the root cause. **Expected effect:** explain which failures the proposed change is expected to prevent.
Do not modify any files, generate implementation details, or propose multiple alternative directions. When the evidence is incomplete, explicitly state the uncertainty and select the most conservative direction supported by the trajectories.

trajectories; a *Reflector*, which extracts strategies and failure patterns; and a *Curator*, which converts these insights into context updates. ACE applies incremental modifications by adding, refining, merging, or removing individual context entries rather than rewriting the entire context at each iteration.

ADAS. ADAS (Hu et al., 2025) denotes Automated Design of Agentic Systems, for which Meta Agent Search is a representative algorithm. It represents an agentic system as executable code and searches over prompts, control flows, component compositions, and interactions among components. The method maintains an archive of previously generated agents and their evaluation results, which a meta agent uses to construct and evaluate new candidates iteratively.

G BENCHMARK DESCRIPTIONS

τ^2-Bench. τ^2-Bench (Barres et al., 2026) evaluates conversational agents in realistic customer-service tasks involving multi-turn dialog, domain policies, and tool-mediated state changes. It adopts a dual-control setting in which the agent and simulated user operate distinct tools over a shared environment. We use the official evaluation harness and report the average task success rate across airline, retail, and telecommunications tasks. The benchmark tests policy reasoning, tool use, information elicitation, and user coordination.

Table 5: The prompt of patch generation of TRUSTEVO.

Patch Generation

You are an expert agent-system maintainer. Your task is to generate the smallest sufficient patch that implements the provided patch direction and addresses the failures demonstrated by the supplied trajectories.

Inputs

Failed-trajectory directory: `<FAILED_TRAJECTORIES_DIR>`
Patch direction: `<PATCH_DIRECTION>`
Trust-region constraints:

- Maximum patch size: `<MAX_PATCH_TOKENS>` changed tokens.
- Maximum number of affected components: `<MAX_AFFECTED_COMPONENTS>`.
- Allowed components or files, when specified: `<ALLOWED_COMPONENTS>`.

Inspect the failed trajectories and the current agent-system implementation. Locate the writable parameters, prompts, configurations, policies, or source files associated with the patch direction, and generate a concrete patch under the following requirements:

1. Implement only the supplied patch direction. Do not introduce unrelated improvements.
2. Modify the smallest possible set of files, parameters, and components needed to address the identified root cause.
3. Do not exceed either the maximum patch-size constraint or the maximum affected-component constraint.
4. When an allowed component or file set is provided, do not modify anything outside that set.
5. Preserve existing interfaces, output formats, and behaviors that are unrelated to the observed failures.
6. Avoid broad refactoring, formatting-only changes, duplicated logic, speculative features, and unnecessary defensive code.
7. Do not modify evaluation scripts, benchmark data, tests, gold answers, or failed-trajectory files.
8. Prefer a localized and directly verifiable behavioral correction over a general rewrite.
9. Ensure that the resulting patch is syntactically valid and can be applied cleanly to the current agent system.

The patch size is measured using only added, deleted, or replaced content; diff metadata and unchanged context are excluded. Components include the planner and independently writable modules such as retrieval, memory, skills, tools, and response policies.
Return only a valid unified diff beginning with file headers such as `---` and `+++`. Do not include Markdown code fences, explanations, summaries, or testing instructions outside the diff.
If no meaningful patch can be generated without violating the trust-region constraints, return exactly:
`NO_VALID_PATCH: <brief reason>`

Terminal-Bench-Hard. Terminal-Bench-Hard (Merrill et al., 2026) evaluates agents on difficult, long-horizon command-line tasks in isolated execution environments. Agents must inspect files, execute commands, modify artifacts, debug failures, and manage intermediate states across realistic workflows. We use the hard subset with the official harness and report the mean task pass rate based on task-specific tests of the final environment state. The benchmark emphasizes planning, tool use, debugging, and recovery.

WebArena-Verified-Hard. WebArena-Verified-Hard (Zhou et al., 2023) is a difficulty-prioritized subset of WebArena-Verified, an audited version of the original WebArena benchmark. Agents complete high-level instructions through browser actions on self-hosted websites, while deterministic and backend-state evaluators verify task completion. We evaluate the 258 hard tasks using the official environment and report the average binary success rate. The benchmark tests grounding, navigation, action sequencing, and persistent-state verification.

Table 6: Structural interventions used to obtain different hyperparameters. The templates differ in both the component being modified and the type of behavior introduced.

Modified Part	**How the Hyperparameter is Changed**
Instructions and behavioral constraints	Change how the agent decomposes tasks, checks intermediate results, formats its output, recovers from refusals, or prioritizes low-cost actions.
Tool interface	Add or modify tool-name validation, argument normalization, required-field checks, and constraints applied before a tool call is executed.
Response parsing	Change how actions are extracted from model responses and how malformed JSON, XML, or action blocks are diagnosed and repaired.
Control loop	Change stopping conditions, timeout handling, repeated-action detection, retry behavior, and the criterion used to determine whether the agent is making progress.
State and policy	Change task routing, memory compression, policy selection, deterministic tie breaking, or the preference for the smallest sufficient action sequence.

Table 7: Experimental results of ablation study of TRUSTEVO using GPT-5.4-NANO. - *Exploration* denotes removing the direction exploration, and - *Trust Region* denotes evolving without trust region.

Method	τ^2	**TB-H**	**WA-VH**
TRUSTEVO	$\mathbf{46.2 \pm 2.2}$	$\mathbf{37.1 \pm 3.4}$	$\mathbf{24.8 \pm 1.8}$
- Exploration	42.8 ± 2.7	33.1 ± 3.8	21.4 ± 2.1
- Trust Region	40.6 ± 3.5	29.9 ± 4.6	18.7 ± 2.8

H OBTAINING DIVERSE HYPERPARAMETERS

The initialization library contains 16 templates covering five structural layers of an agent system. Table 6 summarizes how the templates produce different initial behaviors.

These templates vary along two dimensions. First, they intervene at different locations in the execution pipeline, ranging from natural-language instructions to executable control and state-management logic. Second, they impose different behavioral objectives within the same location. For example, the parser variants emphasize diagnostics, permissive JSON extraction, and XML repair, rather than applying three superficial rewrites of the same prompt.

The resulting systems can, therefore, fail in different ways even before evolution. A strict-format prompt may reduce formatting errors but leave planning failures unchanged. A lenient parser may recover otherwise valid actions without affecting reasoning, and a loop guard may prevent repeated actions while introducing a risk of premature termination. The systems are thus structurally distinct rather than random prompt paraphrases.

I ABLATION STUDY OF TRUSTEVO

Table 7 confirms that both direction exploration and trust-region control contribute consistently across all three benchmarks. Removing exploration decreases the average pass rate by 3.6 percentage points, showing that failure attribution and historical evolution feedback provide more reliable update directions. Removing the trust region causes a larger average drop of 6.3 percentage points and increases the variance across repeated runs, indicating that dynamically constraining patch size and affected components is particularly important for both performance and stability while reducing evolution error and cost. These results also show that the two modules play complementary roles: exploration determines *where* to update, while the trust region controls *how much* to update. A promising direction is therefore to refine the trust-region update mechanism, for example, by adapting the patch-size and component-scope constraints separately according to the confidence and observed gain of each candidate patch.