SUPPRESSING A REJECTED ACTION COSTS A KL: CLOSED-FORM TEST-TIME UPDATES FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are widely deployed as agents that interact with external environments through structured actions such as shell commands, tool calls, or search queries. Once the model emits an action that the environment cannot execute, the interaction step is interrupted. In-place recovery then faces a dilemma: with frozen weights, stateless resampling tends to fall back into similar invalid outputs and repeat the same failure; with weight updates after every rejection, gradient-based adaptation introduces substantial latency and compute cost. We propose TALE (Test-time Adaptation via Likelihood Enforcement). TALE turns an observed rejection into an exact likelihood-suppression target and applies a low-rank representation edit at the model's output side; the update direction is determined analytically and the suppression intensity is fixed by a one-dimensional monotone root-finding procedure, with no backpropagation. The edit reshapes the local probability space and moves subsequent attempts away from the failed neighborhood. Across four benchmarks covering terminal commands, tool calls, and web search, at three model scales, TALE reduces the unresolved rate of interaction steps relative to update-free resampling and saves a large fraction of model calls; relative to gradient-based test-time adaptation methods, it attains comparable or higher task success while cutting end-to-end latency by a factor of several, with negligible additional GPU memory.

1 INTRODUCTION

Large language models (LLMs) are widely used as agents that interact with external environments through shell commands, tool calls, search actions, or executable responses. Once the model emits an output the environment cannot execute, for example a syntax error or an instruction that would damage the environment, the current interaction step is interrupted and the task may fail. To finish the task, the system can only regenerate attempts at the same state after a rejection, until an executable attempt is produced or the budget is exhausted. Existing approaches fall into two classes, each limited on either efficacy or cost (their relation to other lines of work is in Appendix F).

If the weights are left unchanged, the system can put the error or the verifier feedback back into the context, resample at the same state, draw more candidates, or select among them by self-consistency (Shinn et al., 2023; Madaan et al., 2023; Wang et al., 2023; Brown et al., 2024). Such methods need neither a model update nor extra optimization, so a single retry is cheap to implement and to serve. The subsequent samples, however, do not necessarily leave the failure structure: the state already contains a rejected continuation, and resampling can land on the same string, the same AST error, or an invalid action of similar shape, none of which advances the task. When the verifier exposes only a binary, non-executable signal, self-correction without external supervision has even more difficulty crossing the error distribution on its own (Huang et al., 2024; Stechly et al., 2025; Kamoi et al., 2024); in our control experiment (Qwen3-4B), putting the rejected output back into the observation makes the retry candidates more similar to each other rather than less (Appendix E.3). Its cost therefore lies not in a single retry but in the total volume: much of the extra calls is spent on repeated candidates.

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.

The other line changes the model between attempts, including test-time parameter adaptation (Wei et al., 2025; Zuo et al., 2025), weights-side updates (Wang et al., 2026), or local interventions on hidden representations such as inference-time intervention (Li et al., 2023a). Its representatives here are aTTT, whose adapter persists across an episode, and ROSA, which updates the weights for the current task continuously (Wang et al., 2026; Wei et al., 2025); the adaptation unit of both is wider than ours. Compared with frozen-weight retries it can change the next response distribution directly, and therefore move generation away from the failed continuation with fewer model calls. It addresses the distribution-level problem of repeated sampling, but the optimization at the scale of a training step after *every* rejection raises the recovery cost, and the resulting latency can far exceed that of several frozen-weight calls.

For a high-frequency recovery loop a method must change the distribution of the next output without paying what backpropagation or fine-tuning costs: the frozen-weight line is cheap but cannot change behavior, the model-changing line can but lifts the cost to the training scale.

This raises a direct question:

How can an agent succeed in fewer retry turns at a cost close to frozen-weight retries?

The question imposes two requirements. To obtain a successful attempt, the probability of wrong outputs at sampling time must be lowered, and without a model update every regeneration draws from nearly the same distribution, so a weight update is necessary. To avoid backpropagation and candidate regeneration, the update can only be computed from what this failure has already produced, with its effect on the next attempt recomputable on the cache. The two requirements together confine the update to the output side and to a form linear in the parameters, so that it is solved directly rather than optimized iteratively.

We therefore propose **TALE** (Test-time Adaptation via Likelihood Enforcement): it writes the observed rejection as an exactly computable suppression target, solves on the cache for the intensity of a linear displacement, and lowers the probability of the rejected sequences to that target. It serves only the next attempt and is removed once the next attempt is judged.

We compare TALE with two classes of baselines on four agent benchmarks under a shared first attempt and an equal retry budget: update-free resampling, and methods that change the model between attempts. Relative to resampling, TALE lowers the fraction of rejected steps, reduces model calls on most benchmarks and lowers the repetition of subsequent candidates at the same state (Appendix C.2); relative to the weight-update control it attains comparable or higher task success at several times lower end-to-end latency, and it stays on the same order as the adapter-based control (§5.2, Appendix E.4). Task-level readings are reported per benchmark; Figure 1 previews the headline same-budget recovery comparison.

Contributions.

- We propose TALE, a local, optimization-free weight update that changes the output distribution at close to frozen-weight cost, so success comes in fewer turns (§3).
- We derive the suppression target from a KL-regularized objective in closed form (Proposition 1) and show that the achieved reduction decomposes exactly into a linear term and a forward KL drift (Theorem 1) (§4).
- Empirically, on four benchmarks and three model scales, TALE lowers the step rejection rate and reduces model calls, with task success no lower than the matched resampling baseline on most benchmarks; the failures that grammar-constrained decoding leaves behind are syntactically valid but wrong, and those are the ones the edit addresses (§5).

2 Problem Setting

Interaction and step partitioning. We consider the sequential interaction between an agent and an external environment. Let $x \in \mathcal{X}$ denote the current state, comprising the system prompt, the multi-turn dialogue history, previous environmental feedback, and the context needed to synthesize the current action; let $y \in \mathcal{Y}$ denote the action the model generates at state x (e.g., a shell command,

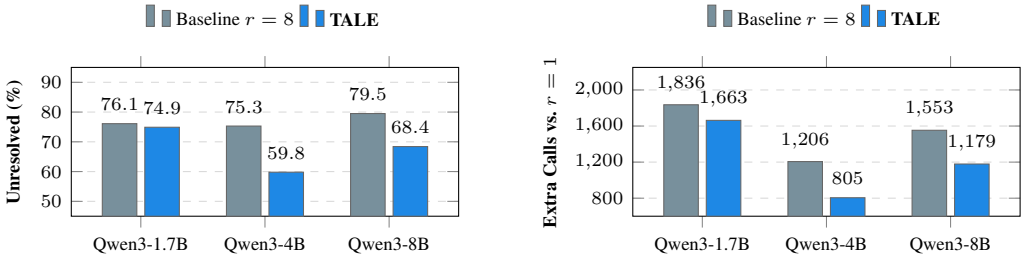

Figure 1: **Same-budget recovery on IC-Bash.** Under $k = 8$ attempts, TALE lowers unresolved steps and extra inference calls relative to update-free resampling across all three model scales (three-seed means; Table 12).

a tool-call structure, or a search query). Before an action enters the real environment and produces persistent state changes, a verifier $\rho : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ checks its validity and executability; on each benchmark, ρ is implemented by that benchmark's intermediate check (§5.1).

We partition the trajectory into **interaction steps** by whether the environmental state advances. *Successful transition*: if an attempt y satisfies $\rho(x, y) = 1$, the step ends successfully, the environment executes y, and the state moves to $x' \sim \mathcal{P}(\cdot \mid x, y)$. *In-place recovery loop*: if the action is rejected ($\rho(x, y) = 0$), the state stays at x and the system retries there; the attempts made at the same state x jointly constitute the recovery loop of a single step.

Under this setting we write $\mathcal{Y}_{\text{acc}}(x) = \{y \in \mathcal{Y} \mid \rho(x, y) = 1\}$ for the acceptable actions at x, and $R^{(j)}_{\text{obs}} = \{y^-_1, \ldots, y^-_j\}$ for the failures accumulated within the step after the j-th attempt is rejected. The update is constructed from the observed rejection history; the complete notation of this subsection is in Appendix B.1.

Policy and edit notation. We write $\pi_0(y \mid x)$ for the base sequence policy and $\pi_A(y \mid x)$ after the step-local output-side edit A. At token position t under prefix $y_{<t}$, $p_{0,t}$ and $p_{A,t}$ are the next-token distributions; $\tilde{h}_t$, E, and P are the cached normalized representation, LM head, and fixed random orthogonal low-rank projection. The exact sequence log-probability is $\ell_A(y \mid x) = \log \pi_A(y \mid x)$, while $\hat{\ell}_A$ is the restricted-support cache score. For an observed rejection y^-_i, g_i is the negative edit gradient at $A = 0$, and δ_i is its target reduction.

Objective and system constraints. Let k denote the maximum attempt budget allowed within a step. If the model first produces an acceptable action at attempt $j \le k$, recovery succeeds and the step consumes $j - 1$ retry turns; if the budget of k attempts is exhausted without an executable action, the step is recorded as an **unresolved step**, and the environment continues with the outcome of the first attempt at this state (no selection among attempts is performed).

Let J denote the number of attempts the step needs,

$$J := \min \left\{ j \le k : y^{(j)} \in \mathcal{Y}_{\text{acc}}(x) \right\}, \tag{1}$$

with the convention $J := \infty$ when all k attempts are rejected (the set is empty), i.e., the step is unresolved and its consumed attempts count as k (we write $\min(J, k)$ wherever that count is used). Within the constrained budget k, we wish to minimize the unresolved probability and the expected attempt consumption:

$$\min_{\text{Policy}} \mathbb{P}(J = \infty) \qquad \text{and} \qquad \min_{\text{Policy}} \mathbb{E}[\min(J, k)]. \tag{2}$$

The step-level and call-level readings through which the experiments measure this objective are defined in §5.1.

3 Algorithm

To achieve this goal we introduce TALE. On a rejection ($r = 0$) it keeps the two artifacts that serving already produces, the rejected token sequences observed at this step and the normalized hidden states

that feed the output layer, recomputes their base probabilities on the cache with the model's own LM head, and turns the observed failures into a single joint update: the minimum-change combination of their suppression directions, with its intensity solved to the target boundary (§3.1–§3.3). The update is applied only if the same recomputation confirms that the retained constraints meet their targets, and it is discarded after the next verifier judgment; Algorithm 1 summarizes it.

Algorithm 1 TALE: Test-Time Adaptation via Likelihood Enforcement

Require: the base policy π_0; the fixed random projection P and the LM head E; the target parameter β, ridge λ, tolerance τ, pruning threshold ϵ; the rejected sequences $y_1^-, \ldots, y_m^-$ of this step, with the state x, their cached base distributions $p_0(\cdot \mid x, y_{<t})$ and normalized representations $\tilde{h}_t$.
Ensure: one step-local edit $A^\star$ for the next attempt, or $A^\star = \emptyset$ (fallback to π_0).
$\mathcal{A} \leftarrow \emptyset$
for $i = 1, \ldots, m$ **do**
  re-score y_i^- from the cache and set its target $\delta_i \leftarrow \delta_i(\beta)$ ▷ one target per rejected sequence
  $g_i \leftarrow \sum_t \left(E^\top u_t\right) \varphi_t^\top$ with $u_t = p_0(\cdot \mid x, y_{<t}) - \mathbf{1}\{\cdot = y_{i,t}\}$, $\varphi_t = P^\top \tilde{h}_t$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$, unless the squared residual of g_i outside $\mathrm{span}\{g_j : j \in \mathcal{A}\}$ is below $\epsilon \|g_i\|_F^2$ ▷ prune near-duplicate failures
end for
repeat
  solve $(K_{\mathcal{A}\mathcal{A}} + \lambda I)\mu_{\mathcal{A}} = \delta_{\mathcal{A}}$ and set $A_{\mathrm{lin}} \leftarrow \sum_{i \in \mathcal{A}} \mu_i g_i$ ▷ $K_{ij} = \langle g_i, g_j \rangle_F$
  $\mathcal{A} \leftarrow \mathcal{A} \setminus \{\arg\min_i \mu_i\}$, if $\min_i \mu_i < 0$
until $\mu_i \geq 0$ for all $i \in \mathcal{A}$, or $\mathcal{A} = \emptyset$
if $\mathcal{A} = \emptyset$ **then**
  return $\emptyset$
else
  $A^\star \leftarrow s A_{\mathrm{lin}}$ with $s = \max_{i \in \mathcal{A}} s_i$, $\hat{\ell}(y_i^- \mid s_i A_{\mathrm{lin}}) = \hat{\ell}(y_i^- \mid 0) - \delta_i$ ▷ bisection on the cache
  return $A^\star$ if $\hat{\ell}(y_i^- \mid A^\star) \leq \hat{\ell}(y_i^- \mid 0) - \delta_i + \tau$ for all $i \in \mathcal{A}$, and $\emptyset$ otherwise ▷ one-sided check; the edit serves only the next attempt
end if

3.1 Target and Direction

The goal of the algorithm is to lower the probability of the observed rejected outputs. Let $y_i^- \in R_{\mathrm{obs}}$ be an observed rejected sequence at this step, with its t-th token written y_t, and let q_i be its base probability, the product of the per-position probabilities before editing. Together with the target parameter β it gives the suppression target

$$\delta_i(\beta) = \frac{1}{\beta} + \log\left(1 - q_i\left(1 - e^{-1/\beta}\right)\right), \tag{3}$$

in nats; since q_i is empirically extremely small (Appendix C), it is almost entirely determined by β. This is the log-probability drop of the KL-regularized optimum for the point-mass negative reward $\tilde{r}_i(y) = -\mathbf{1}\{y = y_i^-\}$ (§4.1); below we only need the drop itself.

Suppressing one sequence cannot act on it alone, and the change should disturb the remaining outputs as little as possible. Unlike a token-level logit bias (logit ban) that acts on tokens regardless of context (Appendix F), the suppression here is driven by the activations of the current state. We therefore define the intervention at the input of the LM head, i.e., on the final-normalized hidden state $\tilde{h}_t$, $\tilde{h} \leftarrow \tilde{h} + AP^\top \tilde{h}$, where A is the update parameter of this step ($A = 0$ reproduces the base policy), P is a fixed random orthogonal low-rank projection (Appendix B.3), and the edited policy is written π_A. This keeps the logit increment $\Delta z_t = EAP^\top \tilde{h}_t$ (E the LM head weight matrix) strictly linear in A, which is the basis of the cached recomputation and of the closed-form solve.

In this coordinate system the direction that lowers a sequence follows from the per-position base distribution $p_0(\cdot \mid x, y_{<t})$ and the token that actually occurred: the residual $u_t(\cdot) = p_0(\cdot \mid x, y_{<t}) - \mathbf{1}\{\cdot = y_t\}$, negative on the rejected token and equal to its base probability elsewhere, is pulled back to the representation space with $E^\top$, multiplied by the projected coordinate $\varphi_t = P^\top \tilde{h}_t$ and accumulated,

$$g_i = \sum_t \left(E^\top u_t\right) \varphi_t^\top. \tag{4}$$

g_i is exactly $-\nabla_A \log \pi_A(y_i^- \mid x)$ at $A = 0$; under the Euclidean metric, the smallest change satisfying the suppression constraint lies along g_i. The algorithm therefore requires $\langle A, g_i \rangle_F \geq \delta_i$ for every observed sequence, providing an analytically controlled sufficient condition for the exact suppression identity; the implementation score and validation step instantiate this condition

on the retained constraints (§4). Since the edit acts only through the output layer, the same A, synthesized from these directions (§3.2), is equivalently a low-rank increment on the output layer, $\Delta E = EAP^\top$, loadable as a runtime delta (§3.4).

3.2 Multi-Sequence Synthesis

Several rejected sequences accumulate at the same state, and one recovery applies only one update, so their requirements are put together as m constraints,

$$\langle A, g_i \rangle_F \geq \delta_i, \quad i = 1, \ldots, m, \tag{5}$$

A new direction that lies approximately in the span of the selected ones very likely comes from an almost repeated failure: the algorithm visits the directions in the order the rejections arrived and prunes those whose squared residual outside that span falls below a threshold relative to their own squared norm (threshold and projection in Appendix B.3); pruned constraints do not enter the later check.

On the retained set $\mathcal{A}$ the algorithm takes the combination with the smallest change, $\min_A \frac{1}{2}\|A\|_F^2$; the solution lies in the span of the directions,

$$A_{\text{lin}} = \sum_{i \in \mathcal{A}} \mu_i g_i, \quad (K_{\mathcal{A}\mathcal{A}} + \lambda I)\mu_{\mathcal{A}} = \delta_{\mathcal{A}}, \quad K_{ij} = \langle g_i, g_j \rangle_F, \tag{6}$$

with K the Gram matrix and $\mu_{\mathcal{A}}$ the coefficients of the constraints, which the KKT conditions require to be non-negative (§4.3); a negative coefficient means that the constraint is not in the active set, so the algorithm drops the most negative one and re-solves, and never puts a discarded constraint back. The stabilizer λ makes the system positive definite under near-collinear directions, so A_{lin} is used as the stabilized direction and the final target achievement is established by the validation step in §3.3; the intensity is solved there. For $|\mathcal{A}| = 1$, $A_{\text{lin}} = \frac{\delta_i}{K_{ii}+\lambda} g_i$, i.e., the minimum-change direction up to the negligible λ.

3.3 Intensity Solving and the One-Sided Check

Given the direction combination whose intensity is not yet fixed, the algorithm solves for the intensity and then validates the update, both on the same recomputation. Let $\hat{\ell}(y_i^- \mid A)$ denote the cache-based approximation to the sequence log-probability under edit A, computed from the rejected tokens, the normalized representations $\tilde{h}_t$, and the model's own LM head. At each position it tracks the highest-probability tokens and the target token; this fixed support is used before and after the edit, while the remaining tokens retain their pre-edit mass in the normalizer. We call this restricted-support score the *implementation score*; the exact full-vocabulary score remains $\ell_A(y_i^- \mid x) = \log \pi_A(y_i^- \mid x)$ and is the object of the theoretical identities. The implementation score keeps the solve at the retry scale, and its effect is measured in Appendix C.

With a single constraint, let $\hat{A} = A_{\text{lin}}/\|A_{\text{lin}}\|_F$ be the unit direction and c the intensity; c is determined by

$$\hat{\ell}(y_i^- \mid c\hat{A}) = \hat{\ell}(y_i^- \mid 0) - \delta_i, \tag{7}$$

where the achieved amount is strictly monotone in the intensity and the root is unique (§4.3), and the linear solution supplies an upper-bound initialization for the target intensity (§4.2); the implementation starts from this intensity, expands the interval outward if necessary, and converges by bisection. With several constraints the retained ones can only share a single scale: multiply A_{lin} by a scalar s, solve the required s_i of each constraint separately and take the largest, which meets all of them with the smallest scale (§4.3); apart from the constraint that needs the largest scale, the remaining constraints are therefore suppressed beyond their own targets.

The final update $A^\star$ ($c\hat{A}$ for a single constraint, sA_{lin} for several) is then validated on the same recomputation, one-sided: the retained constraints must reach their target within the tolerance (Appendix B.3), while deeper suppression is accepted. A passing update is compiled and applied (§3.4); the check covers the finally retained constraints, whose achieved amounts are recorded item by item.

3.4 Execution and Step-Local Lifecycle

In serving, the update obtained from all the erroneous outputs observed in this step is compiled into a low-rank increment on the output layer (§3.1) and loaded; it serves only the next attempt, being removed after the next verifier judgment whether that attempt passes or not, and if that attempt is rejected again with budget left the new rejection is merged into the next update. Once a new state is entered the model restarts from the base policy, and the step ends by the benchmark's native termination rule when the budget is exhausted (§5.1). At most one edit is active at any instant, each edit is cleared at the state transition, and each update corresponds one-to-one with a batch of rejected sequences whose achieved amounts are checked item by item (Appendix C). The resulting update is *step-local*: it is driven by single-step information and serves the next attempt.

4 Theoretical Analysis

This section states the results behind the steps of §3, each followed by the empirical evidence that checks it; the remarks on how the algorithm uses the three results are collected at the end of the section: the suppression target from a KL-regularized objective (§4.1), the *Suppression Cost Identity* (§4.2), and the basis for the minimum-change combination and the one-dimensional intensity solve (§4.3). The derivations and proofs are in Appendix A; the assumptions, the two deviations from the exact form and the failure conditions are in Appendix A.7.

4.1 Origin of the Suppression Target

Proposition 1 (KL-regularized suppression target)**.** *For a rejected sequence y_i^-, let $\tilde{r}_i(y) = -\mathbf{1}\{y = y_i^-\}$ be the point-mass negative reward, and let $\beta > 0$ penalize the deviation from the base policy. The optimizer of the KL-regularized objective multiplies the probability of y_i^- by $e^{-1/\beta}$ and renormalizes the rest, and the resulting drop in sequence log-probability is the target $\delta_i(\beta)$ of §3.1, with $Z_i := 1 - q_i(1 - e^{-1/\beta})$ the normalizer.*

Empirical evidence. Across the four benchmarks the medians of q_i on the retained constraints lie between 2.0×10^{-35} and 4.1×10^{-12}, so the median target $\delta_i(1/16)$ evaluates to 16.000 nats on all four (Table 5; records in Appendix C).

4.2 The Cost of Suppression

The optimizer of §4.1 defines the exact full-vocabulary target, while the deployed edit uses a linear output-side displacement and a cache-based implementation score. The exact policy identity below characterizes the cost of that displacement; the restricted-support score and the final one-sided validation implement the same target at serving time (§3.3, Appendix A.7). For any response y, with π_0 and π_A the policy before and after the edit, write $\delta_A(y) := \log \frac{\pi_0(y|x)}{\pi_A(y|x)}$ for the achieved amount and define the teacher-forced path KL

$$\mathcal{K}_A(y) := \sum_{t=1}^{|y|} \mathrm{KL}\big(p_{0,t}(\cdot \mid x, y_{<t}) \| p_{A,t}(\cdot \mid x, y_{<t})\big).$$

Theorem 1 (Suppression Cost Identity)**.** *If the logit displacement at every position is linear in A, then for any response y*

$$\delta_A(y) = \langle A, g(y) \rangle_F + \mathcal{K}_A(y), \tag{8}$$

where $g(y) := -\nabla_A \log \pi_A(y \mid x)\big|_{A=0}$ is the negative Jacobian at $A = 0$ and $\mathcal{K}_A(y)$ is the accumulated teacher-forced path KL (proof in Appendix A.2).

Empirical evidence. Since the path-KL term is non-negative, the linear solution provides an analytically controlled upper-bound initialization for the target intensity. Root-finding then returns the boundary intensity, and the recorded single-constraint audits show that this intensity stays within the linear initialization (proof in Appendix A.4, with premises stated there). Before root-finding the linear solution had already overshot the target on every recorded constraint, with overshoot medians of 1.01–5.53 nats and maxima of 10.58–55.09 nats; after root-finding the medians of the achievement residuals lie between 1.9×10^{-3} and 0.121 nats (Qwen3-1.7B, three seeds; Appendix C).

4.3 Minimum-Change Allocation and Intensity Solving

This subsection gives the basis for both steps (§3.2, §3.3).

Under the Euclidean metric we minimize $\|A\|_F^2$ subject to $\langle A, g_i \rangle_F \geq \delta_i$ on the retained set $\mathcal{A}$, so the relative importance of the constraints is fixed by the problem itself, with no free weights: the problem is convex with linear constraints (Boyd & Vandenberghe, 2004), and its KKT conditions give exactly the system solved in §3.2, with non-negative coefficients on the strictly active constraints (Appendix A.6).

Once the direction is fixed, the problem becomes one-dimensional. Write $f(c) := \log \pi_{c\hat{A}}(y_i^- \mid x)$ for a rejected sequence y_i^-; because the per-position displacement is linear in c while the normalization is a log-sum-exp of it, f is concave in c, and since the direction suppresses this sequence, $f'(0) = -\langle \hat{A}, g(y_i^-) \rangle_F < 0$. The derivative of a concave function is non-increasing, so f is strictly decreasing on $c \geq 0$: strict decrease gives uniqueness, and the tangent inequality $f(c) \leq f(0) - \langle \hat{A}, g(y_i^-) \rangle_F c$ sends it to $-\infty$, so the root exists and is unique (proof in Appendix A.3). The bisection of §3.3 rests on this monotonicity. The recomputation runs on the restricted support, whose frozen term adds a constant inside the LSE, so the same concavity holds (Appendix A.7).

One recovery applies only one update, so several retained constraints share a single direction and the only adjustable quantity is the overall scale: take $A = sA_{\text{lin}}$. By the same monotonicity, the achieved amount of every retained constraint is strictly increasing in s, so the smallest scale that brings all constraints to their targets is the largest of the scales each constraint requires (proof in Appendix A.5).

Empirical evidence. No intensity that hit the target exceeded the linear solution's own intensity in the single-constraint records, every retained constraint has a normal root status, and no expansion interval had to be widened (Table 4).

Remarks. Read together, the three results are used as follows. The target of §4.1 satisfies $0 < \delta_i(\beta) < 1/\beta$ and tends to $1/\beta$ as $q_i \to 0$, so a single global β needs no per-sequence calibration across benchmarks; equally, among all distributions that lower this sequence to that level, the optimizer is the one closest in KL divergence to the base policy. The path-KL term is non-negative, so the linear constraint $\langle A, g_i \rangle_F \geq \delta_i$ is a sufficient condition for the exact full-vocabulary suppression to reach the target; the implementation uses the cache score and the one-sided validation to transfer this target to the deployed edit. The three operations of §3.2 each follow from one KKT relation of §4.3: the sum of directions from stationarity, the dropping of negative coefficients from $\mu_i \geq 0$, and the zero coefficient of the constraints that are not active. The stabilizer $\lambda > 0$ keeps the system positive definite under near-collinear directions, while the final cache validation establishes target achievement for the retained constraints.

5 Experiments

We evaluate TALE on four benchmarks covering terminal, tool-call and search scenarios, at three local model scales, under a single problem setting and a single protocol. The main line uses $\beta = 1/16$, whose value is justified in Appendix E.2.

5.1 Setup and Benchmark Semantics

Benchmarks and verification semantics. Each benchmark runs its own intermediate check, and except for BFCL v4 single-turn the check asks whether the emitted action can be executed: a shell command that runs, a tool-call batch that parses, a search action that parses into a valid retrieval tag. BFCL v4 single-turn is the exception, judged by the official AST correctness checker, which also requires the call to be the right one. We use all 200 IC-Bash tasks (Yang et al., 2023), all 2,351 single-turn tasks and the 624 multi-turn tasks of BFCL v4 (Patil et al., 2025) that come with complete tool documentation (156 in each of four categories), and the 300 Search-R1 instances (Jin et al., 2025). What each scenario represents and the generation parameters of every benchmark are in Appendix B.1.

Models, arms and comparison protocol. All experiments run on three local Qwen3 models, Qwen3-4B, Qwen3-1.7B and Qwen3-8B (Yang et al., 2025), each evaluated independently under the same problem setting and the same protocol (hardware and serving details in Appendix B.1). We compare three groups of arms: the cost lower bound of a single attempt (**Baseline** $r = 1$), retries without any weight update under the same budget (**Baseline** $r = 8$), and two representatives of the weight-changing line, **ROSA** (Wei et al., 2025) and **aTTT** (Wang et al., 2026). Two further arms appear in the same table: **RA** (Ta et al., 2026), which puts a reviewer model in front of execution, and **Grammar** (Scholak et al., 2021; Dong et al., 2025), grammar-constrained decoding, which removes the actions a static contract rejects; both run under the same verifier as the matched arms, but not under the four rules below. The main comparison of Tables 1 and 2 is run on Qwen3-4B, the model shared with the two weight-changing representatives; the model scale is treated as an ablation, reported at the other two scales in Tables 11 and 12, and the detailed comparison against the two representatives is in Appendix E.4. Baseline $r = 1$ keeps only the shared first attempt and is the reference arm for the unresolved rate and accuracy, with extra calls counted from it; Baseline $r = 8$ retries at the same state without any weight update and with independent attempts (naive retry); TALE may apply a local edit inside a recovery step (§3). Four rules fix the comparison. ***Shared first attempt***: every arm starts from the same first attempt and continues from that attempt's outcome, so a difference between arms comes only from the policy after a rejection. ***Equal budget***: all arms use the same maximum number of attempts. ***Continuation***: a failed step advances by the benchmark's native rule and never selects among attempts. ***Lifecycle***: an edit serves only the next attempt and is cleared after its judgment, and the baselines produce no edit (implementation details in Appendices B.1 and B.3). The resampling baseline stays naive so that the observation is fixed and the difference between arms isolates the policy update; the observation-side control in Appendix E.3 shows why that matters, since writing rejected outputs back into the observation gives **no detectable task-level gain** ($p = 0.50$) and makes the retry candidates more similar to each other.

Readings. The main reading is the unresolved rate, the fraction of a step's triggered failures that stay unresolved after the budget, counted on each arm's own trajectory; we also report step rejection, extra calls versus Baseline $r = 1$, and benchmark accuracy (definitions in Appendix B.2).

5.2 Main Results

In Table 1 the IC-Bash column separates the arms most clearly: neither weight-changing arm turns the same budget into recovery, while TALE is well below update-free resampling, and the same ordering holds at the other two scales (Table 12); TALE is also the lowest on extra calls in the IC-Bash and single-turn columns. RA and Grammar are read differently: RA buys step-level recovery with a reviewer call on every attempt, at several times the calls, while grammar-constrained decoding lowers the unresolved rate only where its contract covers the failing batch and not at all where the verifier judges correctness rather than syntax, since what remains is syntactically valid and still wrong (Park et al., 2024). TALE's target is the rejected sequence itself, not the strings a grammar admits, so it covers both kinds of failure: the two mechanisms are orthogonal and can be stacked.

Baseline $r = 1$ and the per-seed step counts are in Table 11, and the per-seed Search-R1 calls in Appendix E.5; trajectory fragments are in Appendix D.

End-to-end time. Table 2 reports the wall-clock time of the retrying arms. ROSA pays a gradient update on every task and is several times slower than resampling, aTTT's per-update cost is lower, and RA, which pays a reviewer call on every attempt, lands at 1.2–2.6× resampling. TALE is 0.98× the retry baseline on IC-Bash and 1.8–2.6× it on the other three, the price of recomputing an edit at every rejection; the ordering also depends on flow length, so in the short single-step flow aTTT triggers fewer updates and appears cheaper.

Recovery and efficiency. Across model scales TALE moves the step-level recovery curve below update-free resampling on IC-Bash and BFCL v4 multi-turn, while retrying alone changes the task pass rate little: the retry baseline stays level with the single-attempt baseline across scales and benchmarks. TALE improves recovery within the same budget, and its task-level gains concentrate on BFCL v4 single-turn and IC-Bash; on BFCL v4 multi-turn and Search-R1 it stays close to its matched retry baseline, the step-to-task limit summarized in §6 (three-seed readings in Table 12).

Table 1: **Main comparison at seed 0 on Qwen3-4B.** Methods: naive retry (**Baseline** $r = 1$ and **Baseline** $r = 8$), ROSA (Wei et al., 2025), aTTT (Wang et al., 2026), RA (Ta et al., 2026), grammar-constrained decoding (Scholak et al., 2021; Dong et al., 2025), and TALE. Unresolved and accuracy are percentages, extra calls are counts, and an arrow gives the direction in which a metric improves. The second entry in each cell is the change over the reference arm, extra calls over Baseline $r = 8$ and unresolved and accuracy over Baseline $r = 1$ (percent, or points for accuracy), red where it is an improvement and black otherwise; bold marks the best value in a column. With only the shared first attempt, Baseline $r = 1$ has a 100% unresolved rate by construction and an empty extra-call cell; BFCL v4 single-turn is single-step and has no unresolved column, and neither does Search-R1, whose triggered steps are single-digit and too few to report a rate. Three-seed means over the three model scales are in Table 12, the per-seed counts in Table 11.

Method	IC-Bash			BFCL v4 single		BFCL v4 multi			Search-R1	
	Unresolved ↓	Extra calls ↓	Accuracy ↑	Extra calls ↓	Accuracy ↑	Unresolved ↓	Extra calls ↓	Accuracy ↑	Extra calls ↓	Accuracy ↑
Baseline $r = 1$	100.0		27.5		71.5	100.0		**11.2**		33.0
Baseline $r = 8$	71.0 (−29.0%)	1,059	29.0 (+1.5 pp)	4,538	72.9 (+1.4 pp)	92.8 (−7.2%)	2,712	10.9 (−0.3 pp)	10	33.0 (0.0 pp)
ROSA	74.8 (−25.2%)	1,315 (+24.2%)	30.0 (+2.5 pp)	4,507 (−0.7%)	73.2 (+1.7 pp)	92.6 (−7.4%)	2,862 (+5.5%)	10.7 (−0.5 pp)	24 (+140.0%)	33.0 (0.0 pp)
aTTT	80.6 (−19.4%)	1,447 (+36.6%)	28.5 (+1.0 pp)	4,541 (+0.1%)	72.7 (+1.2 pp)	90.7 (−9.3%)	2,828 (+4.3%)	9.0 (−2.2 pp)	**4** (−60.0%)	**33.7** (+0.7 pp)
RA	64.7 (−35.3%)	3,545 (+234.7%)	27.5 (0.0 pp)	9,744 (+114.7%)	72.8 (+1.3 pp)	83.8 (−16.2%)	12,795 (+371.8%)	10.4 (−0.8 pp)	697 (+6870.0%)	**33.7** (+0.7 pp)
Grammar	79.1 (−20.9%)	1,507 (+42.3%)	28.5 (+1.0 pp)	4,844 (+6.7%)	72.8 (+1.3 pp)	87.1 (−12.9%)	**896** (−67.0%)	11.1 (−0.1 pp)	10 (0.0%)	33.0 (0.0 pp)
TALE	**57.3** (−42.7%)	**668** (−36.9%)	**30.5** (+3.0 pp)	**4,190** (−7.7%)	**75.4** (+3.9 pp)	**81.1** (−18.9%)	2,716 (+0.1%)	10.6 (−0.6 pp)	**4** (−60.0%)	**33.7** (+0.7 pp)

Table 2: **End-to-end wall-clock time (Qwen3-4B, seed 0).** Hours of the retrying arms of Table 1; Baseline $r = 1$ keeps the single attempt and grammar-constrained decoding adds no model call of its own, so neither is listed. The second entry is the ratio to Baseline $r = 8$, red where it is below one, and the lowest time in each column is bold. The single-turn column covers all 2,351 tasks, the tasks that pass at the first attempt entering at their measured first-attempt cost; RA's multi-turn figure includes 15 runner errors and its Search-R1 figure is the span of that run's records.

Method	IC-Bash	BFCL v4 single	BFCL v4 multi	Search-R1
Baseline $r = 8$	2.86	**1.90**	**2.04**	**0.11**
ROSA	7.54 (2.6×)	19.03 (10.0×)	12.14 (6.0×)	0.79 (7.2×)
aTTT	4.44 (1.6×)	2.40 (1.3×)	4.29 (2.1×)	0.25 (2.3×)
RA	3.38 (1.2×)	2.84 (1.5×)	4.36 (2.1×)	0.29 (2.6×)
TALE	**2.81** (0.98×)	4.89 (2.6×)	3.58 (1.8×)	0.27 (2.5×)

6 Conclusion and Limitations

TALE turns an observed rejection into a KL-calibrated output-side edit for the next attempt. Its closed-form direction, one-dimensional intensity solve, and step-local lifecycle lower recovery cost within the shared serving path. Across three Qwen3 scales and four benchmarks, TALE lowers unresolved rates on IC-Bash and BFCL v4 multi-turn at every scale, reduces model calls on most benchmarks (up to 33.3%; Table 12), runs 2.7–3.9× faster than ROSA and on the same order as aTTT (Table 2), and adds **near-zero GPU memory**: an 18.86 MiB increment with **no measurable change at load** (Appendix B.4).

The operating scope is summarized below; detailed conditions and audits appear in Appendix A.7.

1. **Step-to-task transmission.** TALE optimizes recovery at the current state; task completion follows each benchmark's credit assignment. We therefore report step-level recovery and task-level accuracy separately (§5.2).
2. **Feedback bandwidth and model access.** TALE uses rejected tokens, cached representations, and the LM head available in instrumentable serving (§3); opaque interfaces require an integration layer.
3. **Model scale.** Evidence covers Qwen3-1.7B, Qwen3-4B, and Qwen3-8B under single-GPU, single-concurrency serving; recoverable room varies with scale (Appendix B.1).
4. **Verifier ceiling.** TALE accelerates entry into the verifier's acceptance region; the verifier defines the attainable accuracy ceiling. Alignment conditions are stated in Appendix A.7.

AI USE STATEMENT

We used generative AI tools for language editing, typesetting, experiment-automation and analysis code, and assistance with mathematical derivations and proofs. The authors reviewed all AI-assisted work, tested the code on the reported runs, verified the derivations against Appendix A, checked every citation against the original paper, and take responsibility for the final content and artifacts.

ETHICS STATEMENT

This work investigates test-time adaptation for autonomous LLM agents executing structured actions, including shell commands and tool invocations. All experiments involving terminal execution (InterCode-Bash) were conducted within strictly isolated, non-networked Docker container sandboxes to prevent arbitrary code execution on host systems or external digital infrastructure. This research does not involve human subjects, personal identifying data, or harmful environmental deployments.

REPRODUCIBILITY STATEMENT

To ensure complete reproducibility of all theoretical and empirical claims: (1) Complete algebraic proofs for Theorem 1 and Propositions 1–5 are fully detailed in Appendix A. (2) Comprehensive serving configurations, vLLM parameters, sampling temperatures, optimization solver hyperparameters, and the sources of the benchmark data together with the filtering steps we applied to them are provided in Appendix B.1. (3) All experiments use the base seeds 0, 10 and 20, the seed of the i-th attempt being the base seed plus i; the comparison of Table 1 is a seed-0 run, the three-seed means $\pm$ population standard deviation over the three scales are in Table 12, and the per-seed counts, the per-seed calls and the single-attempt baseline are documented in Appendix E.5. (4) Following the venue's policy, this work is released under the CC BY 4.0 license.

REFERENCES

Emre Can Acikgoz, Cheng Qian, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. TT-SI: Self-improving LLM agents with test-time training. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.462/.

Nikita Agarwal and Nivedit Jain. FIRE: Failure-informed runtime engineering for reliable language-model agents, 2026. URL https://arxiv.org/abs/2609.26048.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511804441.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, et al. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024. URL https://arxiv.org/abs/2407.21787. arXiv only; no venue.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, et al. DoLa: Decoding by contrasting layers improves factuality in large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=Th6NyL07na.

Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003. doi: 10.1002/rsa.10073.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/5c20ca4b0b20b0bd2f1d839dc605e70f-Abstract-Conference.html.

Zenghao Duan, Liang Pang, Zihao Wei, Wenbin Duan, Yuxin Tian, Shicheng Xu, Jingcheng Deng, Zhiyi Yin, and Xueqi Cheng. Circular reasoning: Understanding self-reinforcing loops in large reasoning models, 2026. URL https://arxiv.org/abs/2601.05693.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, et al. KTO: Model alignment as prospect theoretic optimization. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/ethayarajh24a.html.

Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Di He, Wenhao Huang, and Tianle Cai. In-place test-time training. In *International Conference on Learning Representations (ICLR)*, 2026. doi: 10.48550/arXiv.2604.06169. URL https://arxiv.org/abs/2604.06169.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

Bowen Guan, Zhentao Yin, and Yanming Shen. ParaRecover: A process-level benchmark for error localization and recovery in parallel tool-use agents. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026. URL https://arxiv.org/abs/2609.12345. arXiv:2609.12345.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8137–8153, 2023. URL https://aclanthology.org/2023.emnlp-main.506/.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, 2020. URL https://aclanthology.org/2020.coling-main.580/. 2WikiMultihopQA.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

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. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Shaojie Jiang, Ruqing Zhang, Svitlana Vakulenko, and Maarten de Rijke. A simple contrastive learning objective for alleviating neural text degeneration, 2022. URL https://arxiv.org/abs/2205.02517.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O. Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Proceedings of the Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Yuxuan Jia, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics (TACL)*, 2024. doi: 10.1162/tacl_a_00713.

Tomasz Korbak, Hady Elsahar, Germán Kruszewski, and Marc Dymetman. Controlling conditional language models without catastrophic forgetting. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022. URL https://proceedings.mlr.press/v162/korbak22a.html.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024. URL https://arxiv.org/abs/2409.12917. SCoRe.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, et al. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics (TACL)*, 2019. URL https://aclanthology.org/Q19-1026/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Aristotelis Lazaridis, Aman Sharma, Dylan Bates, Brian King, Vincent Lu, and Jack FitzGerald. When can one neuron fix repetition loops in LLMs?, 2026. URL https://arxiv.org/abs/2606.13705.

Sergey Levine. Reinforcement learning and control as probabilistic inference: A tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018. URL https://arxiv.org/abs/1805.00909.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 41451–41530, 2023a. doi: 10.52202/075280-1797.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, et al. Contrastive decoding: Open-ended text generation as optimization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023b. URL https://aclanthology.org/2023.acl-long.686/.

Ximing Lu, Peter West, Rowan Zellers, et al. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021. URL https://aclanthology.org/2021.naacl-main.338/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023. doi: 10.52202/075280-2019.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi: 10.1007/BF02295996.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 17359–17373, 2022. URL https://openreview.net/forum?id=2QNG8btAqA.

Shuaicheng Niu, Guohao Chen, Yaofo Chen, Zhiquan Wen, Jinwu Hu, Zeshuai Deng, Deyu Chen, Shuhai Zhang, Renjie Chen, Zihao Lian, Shoukai Xu, et al. A survey on self-improving test-time intelligence: Feedback-driven adapting, learning, and scaling at inference, 2026. URL https://arxiv.org/abs/2609.01679. accepted by Machine Intelligence Research.

Işıl Özgü, Yaoxuan Wu, Guy Van den Broeck, and Miryung Kim. The parser already knows: Lightweight bias correction in constrained decoding, 2026. URL https://arxiv.org/abs/2608.10137.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-aligned decoding. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 24547–24568, 2024. doi: 10.52202/079017-0774.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pp. 48371–48392, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):1607–1612, 2010. doi: 10.1609/aaai.v24i1.7727.

Ofir Press, Muru Zhang, Sewon Min, et al. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. URL https://aclanthology.org/2023.findings-emnlp.377/. Bamboogle.

Rafael Rafailov, Archit Sharma, Eric Mitchell, et al. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://openreview.net/forum?id=HPuSIXJaa9.

Amir Mohammad Salehoof, Ali Ramezani, Yadollah Yaghoobzadeh, and Majid Nili Ahmadabadi. A dual-axis taxonomy of knowledge editing for LLMs: From mechanisms to functions, 2025. URL https://arxiv.org/abs/2508.08795.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9879–9894, 2021. URL https://aclanthology.org/2021.emnlp-main.778/.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, et al. S-LoRA: Serving thousands of concurrent LoRA adapters. *arXiv preprint arXiv:2311.03285*, 2023. URL https://arxiv.org/abs/2311.03285. arXiv only; no venue.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023. doi: 10.52202/075280-0377.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=4O0v4s3IzY.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. A contrastive framework for neural text generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 21548–21561, 2022. doi: 10.52202/068431-1566.

Anh Ta, Junjie Zhu, and Shahin Shayandeh. Reinforced Agent: Inference-time feedback for tool-calling agents, 2026. URL https://arxiv.org/abs/2604.27233.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. Let me speak freely? a study on the impact of format restrictions on performance of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, 2024. doi: 10.18653/v1/2024.emnlp-industry.91.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. ♫ MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics (TACL)*, 2022. URL https://aclanthology.org/2022.tacl-1.31/.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023. URL https://arxiv.org/abs/2308.10248.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yanbo Wang, Jinhua Hao, Yuze Shi, Kun Yuan, and Ming Sun. No time like the present: Agentic test-time training for LLM agents, 2026. URL https://arxiv.org/abs/2607.03441.

Chenxing Wei, Hong Wang, Ying He, Fei Yu, and Yao Shu. Test-time policy adaptation for enhanced multi-turn interactions with LLMs. In *MTI-LLM @ NeurIPS 2025*, 2025. URL https://arxiv.org/abs/2509.23166. Workshop on multi-turn interaction; OpenReview venue: MTI-LLM @ NeurIPS 2025 Poster.

Chenxing Wei, Hong Wang, Ying He, Zhongxiang Dai, Bo Jiang, F. Richard Yu, and Yao Shu. Words & weights: Streamlining multi-turn interactions via co-adaptation, 2026. URL https://arxiv.org/abs/2603.01375. ROSA2.

Philipp Emanuel Weidmann, Allen Roush, Judah Goldfeder, Sanjay Basu, and Ravid Shwartz-Ziv. Don't repeat yourself: Stopping verbatim loops at sampling time, 2026. URL https://arxiv.org/abs/2608.22761.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=tAeBzPOvfrx.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, volume 36, pp. 23826–23854, 2023. URL https://openreview.net/forum?id=fvKaLF1ns8.

Zhilin Yang, Peng Qi, Saizheng Zhang, et al. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018. URL https://aclanthology.org/D18-1258/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025. doi: 10.1038/s41586-025-08661-4.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 62138–62160, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to AI transparency. *arXiv preprint arXiv:2310.01405*, 2023. URL https://arxiv.org/abs/2310.01405.

Yuxin Zuo, Kaiyan Zhang, Sheng Li, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025. doi: 10.52202/085713-4376.

A Proofs and Conditions of the Section 4 Claims

This appendix gives the proofs of the claims in §4; the notation agrees with §3 and §4. Here x is the current agent state; π_0 is the base policy and π_A the policy under edit A ($A = 0$ gives π_0); the logits at position t are $z_t(A)$, written $z_t^0 := z_t(0)$ before the edit, and the per-position distribution

is $p_A(\cdot \mid x, y_{<t}) = \mathrm{softmax}(z_t(A))$, written p_0 before the edit; the achieved amount on a response y is $\delta_A(y) := \log \frac{\pi_0(y|x)}{\pi_A(y|x)}$ and the direction is $g(y) := -\nabla_A \log \pi_A(y \mid x)\big|_{A=0}$; the per-position forward KL along y is

$$\mathcal{K}_A(y) := \sum_t \mathrm{KL}\left(p_0(\cdot \mid x, y_{<t}) \,\|\, p_A(\cdot \mid x, y_{<t})\right),$$

and the inner product is $\langle A, B\rangle_F := \mathrm{tr}(A^\top B)$. A rejected sequence is written $y_i^- \in R_{\mathrm{obs}}$, its suppression target δ_i, its direction $g_i := g(y_i^-)$, and the retained constraint set $\mathcal{A}$ (§3.2).

A.1 Proof of Proposition 1 (KL-Regularized Suppression Target)

Proof. Write the objective of Proposition 1 as

$$\max_{\pi(\cdot|x)} \mathbb{E}_{y\sim\pi(\cdot|x)}\big[\tilde{r}_i(y)\big] - \beta\,\mathrm{KL}\big(\pi(\cdot \mid x) \,\|\, \pi_0(\cdot \mid x)\big).$$

It is strictly concave in $\pi(\cdot \mid x)$ over the simplex, so the optimizer is unique. With the point-mass reward, stationarity of the Lagrangian with multiplier $\log Z_i$ gives $\log \pi_i^\star(y \mid x) = \log \pi_0(y \mid x) + \tilde{r}_i(y)/\beta - \log Z_i$, i.e., $\pi_i^\star(y \mid x) = \pi_0(y \mid x) e^{\tilde{r}_i(y)/\beta}/Z_i$, and the normalization constraint yields

$$Z_i = \sum_y \pi_0(y \mid x)\, e^{\tilde{r}_i(y)/\beta} = 1 - q_i + q_i e^{-1/\beta} = 1 - q_i\big(1 - e^{-1/\beta}\big).$$

Since $\tilde{r}_i(y_i^-) = -1$, we have $\pi_i^\star(y_i^- \mid x) = \pi_0(y_i^- \mid x) e^{-1/\beta}/Z_i$; taking logarithms of the ratio gives $\delta_i(\beta) = \frac{1}{\beta} + \log Z_i$. □

The limits quoted in §4.1 follow directly: $Z_i \in (e^{-1/\beta}, 1)$ for $q_i \in (0, 1)$, so $\delta_i(\beta)$ tends to $1/\beta$ as $q_i \to 0$ and to 0 as $q_i \to 1$. The closed form is an established result of KL-regularized objectives (Peters et al., 2010; Levine, 2018; Korbak et al., 2022), used here for a point-mass negative reward in the recovery setting.

A.2 Proof of Theorem 1 (Suppression Cost Identity)

For any position t, by the coordinates of §3.1 the edit changes only the output-side logits, and the displacement is linear in A:

$$z_t(A) = z_t^0 + \Delta z_t(A), \qquad \Delta z_t \text{ linear in } A.$$

Write the per-position log-probability

$$\log \pi_A(y_t \mid x, y_{<t}) = z_t(A)_{y_t} - \mathrm{LSE}\left(z_t(A)\right),$$

and write the difference of the LSE terms as an expectation under the base distribution:

$$\mathrm{LSE}\left(z_t(A)\right) - \mathrm{LSE}\left(z_t^0\right) = \log \sum_v p_0(v \mid x, y_{<t})\, e^{\Delta_t(v)} = \log \mathbb{E}_{p_0}\left[e^{\Delta_t}\right],$$

$$\Delta_t(v) := z_t(A)_v - \left(z_t^0\right)_v.$$

Since Δ_t is linear in A,

$$\delta_A(y) = \sum_t \left\{-\Delta_t(y_t) + \log \mathbb{E}_{p_0}\left[e^{\Delta_t}\right]\right\}.$$

On the other hand, differentiating $\log \pi_A(y \mid x) = \sum_t \left[z_t(A)_{y_t} - \mathrm{LSE}(z_t(A))\right]$ at $A = 0$, with $d_t(v) := \nabla_A z_t(A)_v$ (independent of A by the linearity of the displacement) and $\nabla \mathrm{LSE}(z) = \mathrm{softmax}(z)$:

$$\nabla_A \log \pi_A(y \mid x)\big|_{A=0} = \sum_t \left[d_t(y_t) - \mathbb{E}_{p_0}[d_t]\right]$$

$$\Longrightarrow \quad g(y) = \sum_t \left[\mathbb{E}_{p_0}[d_t] - d_t(y_t)\right].$$

By the linearity of Δ_t, $\langle A, d_t(v)\rangle_F = \Delta_t(v)$, hence

$$\langle A, g(y)\rangle_F = \sum_t \left[\mathbb{E}_{p_0}[\Delta_t] - \Delta_t(y_t)\right].$$

Subtracting the two expressions,

$$\delta_A(y) - \langle A, g(y)\rangle_F = \sum_t \left\{\log \mathbb{E}_{p_0}\left[e^{\Delta_t}\right] - \mathbb{E}_{p_0}[\Delta_t]\right\}.$$

It remains to compute the forward KL per position. Let $H(p_0)$ denote the entropy of the base distribution at position t; from $\log p_0(v) = \left(z_t^0\right)_v - \mathrm{LSE}\left(z_t^0\right)$ we have

$$\mathbb{E}_{p_0}[\log p_0] = -H(p_0) = \mathbb{E}_{p_0}[z_t^0] - \mathrm{LSE}(z_t^0), \qquad \text{i.e.} \qquad \mathrm{LSE}(z_t^0) - \mathbb{E}_{p_0}[z_t^0] = H(p_0).$$

Therefore

$$\begin{aligned}
\mathrm{KL}\left(p_0 \| p_A\right)\big|_t &= \sum_v p_0(v)\left[\log p_0(v) - \log p_A(v)\right] \\
&= -H(p_0) - \mathbb{E}_{p_0}\left[z_t(A)_v\right] + \mathrm{LSE}\left(z_t(A)\right) \\
&= -H(p_0) - \left(\mathbb{E}_{p_0}[z_t^0] + \mathbb{E}_{p_0}[\Delta_t]\right) \\
&\quad + \left(\mathrm{LSE}(z_t^0) + \log \mathbb{E}_{p_0}\left[e^{\Delta_t}\right]\right) \\
&= \underbrace{\left(\mathrm{LSE}(z_t^0) - \mathbb{E}_{p_0}[z_t^0] - H(p_0)\right)}_{=0} + \log \mathbb{E}_{p_0}\left[e^{\Delta_t}\right] - \mathbb{E}_{p_0}[\Delta_t],
\end{aligned}$$

whose right-hand side has, position by position, the same summands as the expression obtained by subtracting the two equations above. Summing over t gives

$$\delta_A(y) = \langle A, g(y)\rangle_F + \mathcal{K}_A(y). \qquad \square$$

By Jensen's inequality, $\log \mathbb{E}_{p_0}[e^{\Delta_t}] \geq \mathbb{E}_{p_0}[\Delta_t]$, so $\mathcal{K}_A(y)$ is non-negative. This path-KL term quantifies the distributional movement induced by the output-side edit.

A.3 Concavity and the Existence and Uniqueness of the Root

Proposition 2. *Fix a rejected sequence y^- and a direction $\hat{A}$, and let $f(c) := \log \pi_{c\hat{A}}(y^- \mid x)$. If the logit displacement at every position is linear in A, and the direction suppresses this sequence ($\langle \hat{A}, g(y^-)\rangle_F > 0$), then f is concave and strictly decreasing on $[0, \infty)$.*

Proof. $f(c) = \sum_t \left[z_t(c\hat{A})_{y_t} - \mathrm{LSE}(z_t(c\hat{A}))\right]$. By the linearity of the displacement, $z_t(c\hat{A})$ is an affine function of c, so the first term is affine; the second is the composition of the convex function LSE with an affine map and is therefore convex in c. An affine function minus a convex function is concave, and a finite sum preserves concavity. Moreover,

$$f'(0) = \left\langle \hat{A}, \nabla_A \log \pi_A(y^- \mid x)\big|_{A=0} \right\rangle_F = -\left\langle \hat{A}, g(y^-) \right\rangle_F < 0,$$

and the derivative of a concave function is non-increasing, so $f'(c) \leq f'(0) < 0$ for all $c \geq 0$ and f is strictly decreasing on $[0, \infty)$. $\square$

Corollary 1 (Existence and uniqueness of the root)**.** *Strict decrease gives uniqueness. By concavity and the tangent inequality $f(c) \leq f(0) - \langle \hat{A}, g(y^-)\rangle_F c$, we have $\lim_{c\to\infty} f(c) = -\infty$, so the equation $\delta_{c\hat{A}}(y^-) = \delta_i$ has exactly one root in $c > 0$. Degenerate directions ($f'(0) = 0$, for instance a direction orthogonal to the output-layer weights that changes no logits) do not satisfy the premise of the proposition and are outside the above range. The implementation starts from the intensity of the linear solution, expands the interval outward if necessary to locate the end below the target level, and then converges by bisection (§3.3); a failure to close can only come from numerical causes, in which case the algorithm abandons that update, and the measured root status is normal in all records (Appendix C).*

A.4 The Linear Solution and the Upper Bound on the Intensity

Proposition 3. *Under a single constraint, with the exact direction g_i and without the stabilizer:*

1. $\langle A, g_i\rangle_F \geq \delta_i \implies \delta_A(y_i^-) \geq \delta_i$;
2. *along the unit direction $\hat{A} = g_i/\|g_i\|_F$, the intensity that hits the target satisfies $c^\star \leq c_{\text{lin}} = \delta_i/\langle \hat{A}, g_i\rangle_F$.*

Proof. Write $h(c) := \delta_{c\hat{A}}(y_i^-)$. By §A.3, f is strictly decreasing, so $h(c) = f(0) - f(c)$ is strictly increasing. (1) follows from the identity of Appendix A.2 together with the non-negativity of the KL term. (2) At $c = c_{\text{lin}}$ we have $\langle c_{\text{lin}}\hat{A}, g_i\rangle_F = \delta_i$, so $h(c_{\text{lin}}) \geq \delta_i$ by part (1); since $h(c^\star) = \delta_i$ and h is strictly increasing, $c^\star \leq c_{\text{lin}}$. □

If the KL term at c_{lin} is strictly positive (as it generally is under a last-layer low-rank edit), then $h(c_{\text{lin}}) > \delta_i$, i.e., the linear solution has already overshot the target: the role of root-finding is to retract the intensity from c_{lin} back to $c^\star$, not to correct an error. This bound holds only for a single constraint, with the exact direction and without the stabilizer; with the stabilizer or with several constraints the guarantee passes to the one-sided check of §3.3 (§4.2).

A.5 Maximum Scale for Multiple Constraints

Proposition 4. *Given a common direction A_{lin} and a set of retained constraints $\mathcal{A}$, let $h_i(s) := \delta_{sA_{\text{lin}}}(y_i^-)$ be the achieved amount of the i-th constraint at scale s, and let s_i be the scale at which it hits its target ($h_i(s_i) = \delta_i$). Taking $s = \max_{i\in\mathcal{A}} s_i$, every retained constraint satisfies $\delta_{sA_{\text{lin}}}(y_i^-) \geq \delta_i$.*

Proof. First confirm that every retained constraint satisfies the premise of Appendix A.3. By the KKT relation of §4.3 (without the stabilizer, see Appendix A.6), for the strictly active constraints $K_{\mathcal{A}\mathcal{A}}\mu_{\mathcal{A}} = \delta_{\mathcal{A}}$, so $\langle A_{\text{lin}}, g_i\rangle_F = \sum_j \mu_j \langle g_j, g_i\rangle_F = \delta_i$; for the remaining constraints $\mu_i = 0$, and feasibility gives $\langle A_{\text{lin}}, g_i\rangle_F \geq \delta_i$. Both cases give $\langle A_{\text{lin}}, g_i\rangle_F \geq \delta_i > 0$, hence $A_{\text{lin}} \neq 0$. The unit direction $\hat{A} = A_{\text{lin}}/\|A_{\text{lin}}\|_F$ therefore satisfies $\langle \hat{A}, g_i\rangle_F > 0$, and writing $s = \|A_{\text{lin}}\|_F\, c$ reduces the statement to the unit-direction case of Appendix A.3; by that result each h_i is strictly increasing in s. Hence $s \geq s_i \Rightarrow h_i(s) \geq h_i(s_i) = \delta_i$ for every $i \in \mathcal{A}$. □

Consequently, apart from the constraint that needs the largest scale, the remaining constraints are suppressed more deeply than their own targets; the achievement residuals of Appendix C are recorded item by item on exactly this fact.

A.6 Minimum-Change Combination and the Role of the Stabilizer

Proposition 5. *Given the set of retained constraints $\mathcal{A}$, the convex problem*

$$\min_A \frac{1}{2}\|A\|_F^2 \quad \textit{s.t.} \quad \langle A, g_i\rangle_F \geq \delta_i, \qquad i \in \mathcal{A}$$

has an optimizer of the form $A_{\text{lin}} = \sum_{i\in\mathcal{A}} \mu_i g_i$ with $\mu_i \geq 0$; for the strictly active constraints, $K_{\mathcal{A}\mathcal{A}}\mu_{\mathcal{A}} = \delta_{\mathcal{A}}$, where $K_{ij} = \langle g_i, g_j\rangle_F$.

Proof. Write the Lagrangian

$$\mathcal{L}(A, \mu) = \frac{1}{2}\|A\|_F^2 - \sum_{i\in\mathcal{A}} \mu_i \left(\langle A, g_i\rangle_F - \delta_i\right), \qquad \mu_i \geq 0.$$

Stationarity in A gives $A = \sum_{i\in\mathcal{A}} \mu_i g_i$; complementary slackness sets $\mu_i = 0$ for the constraints not taken with equality; taking equality for the strictly active constraints gives $K_{\mathcal{A}\mathcal{A}}\mu_{\mathcal{A}} = \delta_{\mathcal{A}}$. If a negative coefficient appears in the solution, that constraint does not satisfy $\mu_i \geq 0$ and is not in the active set, and §3.2 removes it and re-solves accordingly; the coefficient of a non-active constraint is zero, so it does not enter the solution. □

The role of the stabilizer. The implementation solves $(K_{\mathcal{A}\mathcal{A}} + \lambda I)\mu_{\mathcal{A}} = \delta_{\mathcal{A}}$, which differs from the unregularized KKT equality by $\lambda\mu_{\mathcal{A}}$; this step therefore carries no achievement guarantee by itself. When $K_{\mathcal{A}\mathcal{A}}$ is close to singular, $\lambda > 0$ makes the system positive definite with a unique solution, which is purely a numerical stabilization (§3.2). The final guarantee comes from the check of §3.3, and covers only the finally retained constraints, not those removed by near-collinearity or by a negative coefficient.

A.7 Assumptions, Approximations, and Failure Conditions

This subsection lists the conditions underlying §4.1–§4.3 and points out when the method does not produce an effective recovery.

The conclusions of §4.1–§4.3 rest on two structural conditions. First, the update acts only on the output side (an edit of the output-side representation at the last layer), so the logit displacement at each position is linear in the update parameter A; both the identity of §4.2 and the concavity of §4.3 follow from this. Second, the tokens of the rejected sequence and the normalized representations from its generation can be retained for the local recomputation; hence the update needs neither to re-run the model nor to backpropagate. In addition, the generation of the candidates and the later recomputation satisfy a protocol condition: the same tokenization and the same decoding configuration, otherwise the sequence probabilities of the two are not comparable. The monotonicity of §4.3 also requires the linear term of the direction to be positive for the constraint: with the single-constraint direction this holds by construction, since the direction is the normalized g_i and its linear term is $\|g_i\|_F > 0$, whereas in the multi-constraint solve a constraint can in principle be orthogonal to the combined direction, in which case scaling that direction does not move it toward its target and the one-sided check reports it as under-suppressed.

Beyond these conditions, the paper deviates from the exact form in two places. The first is that the recomputation of sequence probabilities and the root-finding both run on a restricted support: at each position the tokens of highest probability and the target token are kept, the displacement of the remaining tokens is taken as zero, and their mass still enters the normalizer; its measured effect is in Appendix C. The second is that the output-layer edit is confined to the 32-dimensional coordinates given by a fixed random orthogonal projection (Appendix B.3); this restricts the space of directions that can be changed, and the deviation it introduces is covered, together with the previous approximation, by the check of §3.3. The stabilizer and the two exclusion rules of §3.2 (pruning near-collinear constraints, removing negative-coefficient constraints) are not part of the guarantee above: they only decide which set of constraints enters the solve, and achievement is decided by the check of §3.3 on the finally retained constraints.

There are two situations in which the conclusions above do not apply or do not produce an effective recovery. First, the verifier's judgment disagrees with the suppression target: if it accepts erroneous outputs or rejects correct ones, the suppression direction points at the wrong continuation. Second, an improvement at the step level does not automatically translate into task-level success: the downstream interaction and the credit assignment are decided by the benchmark, and we report those readings separately.

Of the two, step-level transmission has corresponding readings in §5 (the layered step-level and task-level readings of §5.2); the disagreement between the verifier and the suppression target cannot be tested with our data (the verifier is provided by the benchmark) and is listed in §6.

B Protocol and Settings

B.1 Setup

Hardware and environment: each of the three local models is served on a single NVIDIA GeForce RTX 4090 (24,564 MiB) with vLLM 0.23.0 (Kwon et al., 2023) (PyTorch 2.11.0+cu130, CUDA 13.0) under single-concurrency serial evaluation; the drivers are 590.48.01 (Qwen3-4B), 580.173.02 (Qwen3-1.7B) and 590.48.01 (Qwen3-8B). All four benchmarks use non-thinking generation, and the three scales use the same family of runners and the same sampling settings (values below). The serving configuration of Qwen3-8B matches the other two scales (single concurrency, `max_model_len` 20,480, a resident updater, LoRA target set to the LM head).

- **InterCode IC-Bash.** Generation: temperature 0.7, top-p 0.95, max tokens 512, max steps 10. Retry when a command cannot execute, at most 8 times.
- **BFCL v4 single-turn.** Generation: temperature 0.7, top-p 0.8, top-k 20, repetition penalty 1.0, presence penalty 1.5, max tokens 16,384. Retry on a wrong answer, at most 8 times.
- **BFCL v4 multi-turn.** Generation: the OpenAI tools interface, the official tool-message context policy (tool results are written back as official tool messages), max new tokens 2,048, at most 20 tool rounds per turn, concurrency 1; sampling temperature 0.7, top-p 0.8, top-k 20, min-p 0, repetition penalty 1.0, presence penalty 1.5. Only a genuinely invalid tool-call batch triggers a step-level retry, at most 8 times; no-tool or empty responses are handled as the official turn stop and do not trigger a retry.
- **Search-R1.** Generation: temperature 0.7, top-p 0.8, max tokens 1,024, at most 3 search rounds per task, each retrieval returning the top-3 passages (wiki-18 BM25). Retry only when the action is invalid (unparseable), at most 8 times; wrong answers and valid searches do not trigger a retry, and tasks are judged by SubEM.

Benchmark semantics.

- **InterCode-Bash (Terminal)** (Yang et al., 2023): terminal operation is a frequent need of agents in coding and computer use, and it is extremely sensitive to syntax, where a small change alters execution semantics. Each task states a shell goal; the agent emits commands step by step in an isolated environment, and the environment returns the execution result and the new state. We use all 200 tasks, and the intermediate verifier $\rho(x, y)$ accepts a synthesized command only if it executes without syntax errors or non-zero exit codes.
- **BFCL v4 Single-Turn (Tool-Calling)** (Patil et al., 2025): such operations require the model to emit tool names in a prescribed format with the arguments filled in, while the tools and their usage lie outside the model's prior knowledge. The set evaluates JSON-formatted tool use across all 2,351 official tasks, and the verifier is the official Abstract Syntax Tree (AST) checker, which validates function names, mandatory parameters and argument type constraints.
- **BFCL v4 Multi-Turn (Tool-Calling)** (Patil et al., 2025): tasks consist of several user turns in which the agent synthesizes a batch of tool calls, the environment executes it and writes the result back into the next turn's context until the trajectory ends. We evaluate on the 624 tasks with complete documentation and stable tool execution environments across four diverse categories (156 tasks each); the remaining 176 tasks of the official inventory involve tools whose documentation is incomplete in our environment. The verifier triggers recovery exclusively when a synthesized batch is structurally invalid or unparseable.
- **Search-R1 (Information Retrieval)** (Jin et al., 2025): the model obtains information through a search tool and then answers or continues to search, with the environment returning a few retrieved passages. The 300 instances are local multi-hop search-agent tasks drawn from locally available open-domain QA data (NQ, HotpotQA, 2WikiMultihopQA, MuSiQue and Bamboogle (Kwiatkowski et al., 2019; Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022; Press et al., 2023)). The verifier checks whether the emitted string parses into a valid retrieval action tag (`<search>...</search>`), and task success is Substring Exact Match (SubEM).

Implementation of the comparison protocol. On all four benchmarks the first attempt is shared, and each arm continues from the same outcome when it passes or from the same state and the same rejected output when it is rejected; the maximum attempt budget is identical across arms; in multi-step tasks the state advances by the benchmark's native rule, a step that exhausts its budget continues with the outcome of its first attempt, and no arm selects among attempts. The TALE edit is cleared after the next verifier judgment, the baselines produce no edit, and the scope and clearing rules of the two control methods follow Appendix E.4.

Sets at a state. For a state x the action space $\mathcal{Y}$ is conceptually partitioned into three mutually exclusive subsets: the *acceptable set* $\mathcal{Y}_{\mathrm{acc}}(x) = \{y \in \mathcal{Y} \mid \rho(x, y) = 1\}$; the *observed rejected set* $R^{(j)}_{\mathrm{obs}} = \{y_1^-, \ldots, y_j^-\}$, the failures accumulated within the step after the j-th attempt is rejected,

where y_m^- is the action of attempt m and $\rho(x, y_m^-) = 0$; and the *unobserved failures*, the actions that ρ would reject but that have not been observed at this step. When deciding the next attempt, the only factual information available is the accumulated rejection history $R_{\text{obs}}^{(j)}$ together with the forward activations retained during generation. The guarantee of §4 covers only the observed, retained constraints (Appendix A.7), so the unobserved failures cannot be enumerated by the update, and the effect on them is read only through the step-level and task-level measurements of §5.2.

B.2 Readout Conventions

The numerator and denominator of the main readings are counted at the step level on each arm's own trajectory (§5.2). A step is *rejected* when at least one of its candidates was rejected, so the *step rejection rate* is the fraction of executed steps that were rejected; the *unresolved rate* of the main reading is its conditional form, the fraction of rejected steps that remained unresolved after the budget. The definitions of the *step*, of *triggering a retry* and of *being unresolved* are as follows.

- **InterCode IC-Bash**: a step is one chain of command attempts at the same state; a step is triggered when at least one of its candidates was rejected; it is unresolved when no executable command exists after the budget is exhausted.
- **BFCL v4 single-turn**: a single-step benchmark; this column coincides with $1 - \text{Accuracy}$ and is not reported.
- **BFCL v4 multi-turn**: a step is one round within the same user turn; a step is triggered when an invalid tool-call batch appeared in the round; it is unresolved when the round still produced no valid tool-call batch after the budget was exhausted.
- **Search-R1**: a step is one chain of action attempts; a step is triggered when the action of the step is invalid (unparseable); it is unresolved when no executable action exists after the budget is exhausted.

Seed convention: the base seeds are 0, 10 and 20, and the seed of the i-th attempt is the base seed plus i.

B.3 Solver Parameters

TALE evaluator and solver. The top-64 tokens per position together with the target token are kept (the treatment of the remaining tokens is the deviation of Appendix A.7); at most 16 rejected sequences enter an update; the scoring window is 5,120 tokens; the near-collinearity pruning threshold is 10^{-6}; the stabilizer is $\lambda = 10^{-8}$; the projection rank of the last-layer edit is 32 (a fixed random orthogonal projection, generated with random seed 20260830); the bisection starts from the linear solution, runs at most 12 rounds with root tolerance 10^{-6}; the check is one-sided (relative tolerance 10^{-5}, absolute tolerance 10^{-6}; under-suppression is recorded as a violation, over-suppression is not); the intensity is uniquely determined by root-finding, and no additional norm clipping is applied to the update. All of the above agree item by item across the three scales.

Choice of the low-rank projection. P is a fixed random orthogonal projection generated once (rank 32, random seed 20260830) and reused across updates and across tasks; the edit is therefore the composition of "project onto 32 coordinates and map back to the output side with A" (§3.1). Two reasons support a random orthogonal projection over a commonly used data-dependent basis (such as an SVD/PCA over calibration representations). First, it is data-independent: the projection can be fixed in advance and reused directly, without collecting and factorizing a batch of calibration representations offline, which matches the design goal of keeping the update at the retry scale (§3). Second, it does not systematically prefer any direction: Johnson–Lindenstrauss type results (Dasgupta & Gupta, 2003) show that a random projection approximately preserves inner products and distances of the projected vectors, so the residual and the outer product in the low-dimensional coordinates still correspond to the local geometry in the representation space; the low rank at the same time confines the edit to these 32 coordinates, and together with the minimum-change combination of §3.2 the degrees of freedom of the edit do not grow as rejected sequences accumulate. Note that distance preservation is only the rationale for this choice, not a per-update guarantee: the deviation it introduces is covered, together with the other deviation of §A.7, by the check of §3.3, and whether an update meets its target is decided item by item on the real recomputation.

Settings of the two weight-changing controls. ROSA uses weights-only updates (it does not intervene on the hidden states) and takes the $\beta = 1$ setting its paper requires; the adaptation unit of the original paper is one task, whereas the multi-step benchmarks here do not assume that a task is solvable in one step, so we let one step correspond to one task of the original paper and clear its update at the end of the step (whether it succeeded or not), following the original task-level lifecycle. aTTT carries its adapter for the whole episode. Apart from these two variants and settings, both follow the default configuration of their public implementations with no additional tuning; both were run on Qwen3-4B, and their readings are reported alongside the two resampling baselines and TALE (§5.2, Appendix E.4).

B.4 GPU MEMORY

The edit of TALE is compiled into a low-rank increment on the output layer and enters the same serving through the existing low-rank adaptation loading path (§3.1, §3.4), so its additional memory is only this increment itself: the backbone, the LM head and the KV cache are shared with the update-free service, and at most one edit is alive at any instant, so this overhead does not accumulate with the number of retries or the episode length (modern serving stacks have mature support for hot-swapping low-rank increments and adapters (Sheng et al., 2023)).

On our serving configuration (single card, single concurrency) we ran one "load → generate → unload" probe for each of 4 low-rank increments produced by the official runs ($\beta = 1/16$), summarized in Table 3.

Table 3: **Hardware and memory footprint profiling.** Single RTX 4090 running Qwen3-4B under vLLM with rank-32 TALE edits.

Profiling Metric	**Measurement**
Base serving VRAM footprint	12,144 MiB (of 24,564 MiB)
One low-rank increment (file)	18.86 MiB
Change at load, whole card / service process	0 MiB
Change after generation	at most 4 MiB

Loading the increment produced **no observable change** in either the whole-card or the service-process memory readings, and generation added at most 4 MiB. That is, apart from this existing low-rank adaptation loading path, TALE introduces **no observable additional memory footprint**.

C APPROXIMATIONS AND MEASURED ACHIEVEMENT

This appendix checks two things: the restricted support on which the recomputation and the root-finding of §3.3 rest (the deviation of §A.7 that needs to be measured separately), and the conclusions of §4.2 and §4.3 about the achieved amounts and the scales.

Readout definitions. All readings are taken from the logged update records of Qwen3-1.7B over three seeds (the same batch of trajectories as the TALE rows at Qwen3-1.7B in Table 11). Each update logs one record; the constraints that entered the solve of that update are its *recorded constraints*, and those that still carry a root record after the solve and the check are its *retained constraints*; the difference between the two is what the two exclusion rules of §3.2 (the near-collinearity pruning and the removal of negative coefficients) removed. The achievement residual is defined as $|\text{achieved} - \delta_i|$ in nats, computed item by item on the retained constraints; the root status is taken from the root record of each retained constraint. Table 4 summarizes these readings over the four benchmarks.

Achieved amount. The medians of the residuals are only 1.2×10^{-4}–7.6×10^{-3} times the suppression target ($\delta_i(1/16) \approx 16$): root-finding retracts the linear solution back to the target boundary. The largest residuals concentrate on updates in which several constraints share one direction: the common scale takes the largest of what the constraints require, so the remaining constraints are suppressed more deeply than their own targets (Appendix A.5); this is the same fact, seen from the other side, as the one-sided rule of §3.3 (under-suppression recorded as a violation, over-suppression not). In the logged updates, the records with under-suppression beyond tolerance on retained constraints

Table 4: **Achieved amounts and root status on the logged updates (Qwen3-1.7B, three seeds).** Residuals are in nats and computed on the retained constraints.

Benchmark	Updates	Recorded / Retained	Residual med. / max	Intensity $\leq$ linear	Root status
InterCode IC-Bash	5,091	6,854 / 6,461	2.9×10^{-3} / 11.67	4,257/4,257	6,461/6,461 normal
BFCL v4 single-turn	15,150	26,086 / 24,327	5.0×10^{-3} / 19.97	8,824/8,824	24,327/24,327 normal
BFCL v4 multi-turn	1,177	1,319 / 1,298	1.9×10^{-3} / 3.80	1,107/1,107	1,298/1,298 normal
Search-R1	1,302	3,809 / 3,420	0.121 / 7.13	424/424	3,420/3,420 normal

and the root records that failed to locate the boundary are both zero (the few under-suppressed records all fall on removed constraints and are not subject to the check); updates abandoned by §3.3 are not logged, and are therefore outside these readings.

Upper bound. Among the single-constraint records, no intensity that hit the target exceeded the intensity of the linear solution itself: 4,257/4,257, 8,824/8,824, 1,107/1,107 and 424/424 over the four rows, consistent with the conclusion of §4.2 (Appendix A.4; that conclusion assumes an exact direction, no stabilizer and a single constraint, and the counting here follows the same premise item by item).

Overshoot. The linear solution had already overshot the target on every recorded constraint of the four benchmarks (§4.2): the medians of the overshoot lie between 1.01 and 5.53 nats and the maxima between 10.58 and 55.09 nats, differing across the four benchmarks. This is the amount root-finding retracts, and it also gives the practical magnitude of the KL term in the identity of §4.2.

Coverage. Constraints removed for near-collinearity or for a negative coefficient do not enter the retained set and are not checked; their share of the recorded constraints is 5.7% (IC-Bash), 6.7% (single-turn), 1.6% (multi-turn) and 10.2% (Search-R1), i.e., the check of this appendix covers 89.8%–98.4% of the recorded constraints. This agrees with §A.7: the two exclusion rules only decide which group of constraints takes part in the solve.

Effect of the restricted support. Its effect on sequence scoring is of the same order as the bisection resolution: the medians of `root_abs_error` are 1.9×10^{-3}–2.8×10^{-3} over the four rows, which corresponds to the resolution of 12 bisection rounds on the current interval rather than to an achievement error (the root tolerance is 10^{-6}, and whether a target is reached is decided separately by the check of §3.3). All retained constraints have a normal root status, and no interval ever had to be expanded outward. The diagnostic field `l0_sparse_minus_full` records an overall offset between the serving side and the cached recomputation (medians of about 73–82 nats over the four rows); it mixes different numerical conditions on the two sides and is not used as an error bound for this approximation.

Source of the records. All four rows come from the official runs of Qwen3-1.7B with three seeds (the same trajectories as the TALE rows at Qwen3-1.7B in Table 11, with the same evaluator parameters and attempt budget); the per-update records are taken from the logged update records of that run, while the per-task numbers of updates, triggers and unresolved steps are recorded separately in a ledger, the two agreeing item by item. The medians of q_i and δ_i below are taken from the same batch of records.

C.1 Suppression Targets

On all four benchmarks what is suppressed are extremely low-probability sequences: the medians of q_i (Table 5) lie between 2.0×10^{-35} and 4.1×10^{-12}, far below 1. Hence in $\delta_i(\beta) = \frac{1}{\beta} + \log\left(1 - q_i\left(1 - e^{-1/\beta}\right)\right)$ the term depending on q_i is negligible, and the median $\delta_i(1/16)$ reaches $1/\beta = 16$ on all four rows: the suppression level is determined by β alone and is approximately independent of which rejected sequence it is (§4.1). The provenance of each row and the item-by-item readings are in the first part of this appendix.

Table 5: **Empirical likelihood audit.** Medians of the base sequence probability q_i and of the resulting suppression target over the retained constraints (Qwen3-1.7B, three seeds).

Benchmark	**Median** q_i	**Median** $\delta_i(1/16)$
InterCode IC-Bash	4.1×10^{-12}	16.000
BFCL v4 single-turn	1.1×10^{-15}	16.000
BFCL v4 multi-turn	5.5×10^{-17}	16.000
Search-R1	2.0×10^{-35}	16.000

C.2 Repetition of Retry Candidates

The claim of §1, that the update lowers the repetition of subsequent candidates at the same state, can be measured directly. On BFCL v4 single-turn with Qwen3-1.7B the two arms share the first attempt and use the same seed for each attempt, with every request identical word for word except for the model identity, so a difference can only come from the update; we take the tasks in which both arms used all 8 attempts (2,075 task instances over three seeds, each retry containing 7 candidates) and compare them in a paired way. Counting two candidates as equal when their normalized canonical tool calls are exactly equal, the pairwise repetition rate of the retry candidates is 0.767 for TALE against 0.882 for Baseline $r = 8$ (paired difference -0.114, 95% interval $[-0.131, -0.098]$ clustered by task); the fraction of candidates identical to the first rejected output is 0.763 against 0.879 (-0.116, $[-0.132, -0.100]$); Table 6 collects the readings. Comparing only the sequence of function names, both arms are above 0.97 (0.979 against 0.989): what the update changes is mainly the content of the call (arguments and formatting), not which function is called. This agrees with the goal of §4.1—what is suppressed is the probability of the observed sequence itself, so what loses repetition is the rejected output rather than the choice of function.

Table 6: **Candidate diversity audit (Qwen3-1.7B, BFCL v4 single-turn).** Paired comparison on the 2,075 task instances where both arms used all 8 attempts; brackets give 95% intervals clustered by task.

Metric (canonical tool call)	**Baseline** $r = 8$	**TALE** ($\beta = 1/16$)
Pairwise candidate repetition rate	0.882	**0.767** $[-0.131, -0.098]$
Identical to the 1st rejected output	0.879	**0.763** $[-0.132, -0.100]$
Function-name sequence repetition	0.989	0.979

D Qualitative Cases

This appendix gives four trajectory fragments that illustrate how the update changes the next attempt after a rejection. They serve as an illustration of the mechanism only, and the overall readings are those of §5.2. All four are from seed 0; the first three are from Qwen3-4B and the fourth from Qwen3-1.7B.

D.1 InterCode IC-Bash: from an unexecutable pipeline command to an executable command

Task `bash:fs_3:050`. Both arms share the same first attempt, which emits the command

```
df -h /workspace | awk '{print $5 " " $ $1}' | grep -E '^([0-9]+%)' | cut -d'%' -f1 | cut -d'
    ' -f2 | tr -d ' '%'
```

in which `awk` writes the field number as `$ $1` (so what is fetched is the field content); the command cannot execute, and the first attempt of this step is therefore rejected.

After 2 updates at this step, the TALE arm gives at its 3rd attempt

```
df -h /workspace | awk '{print $5}' | cut -d'%' -f1 | sed 's/[^0-9]//g' | xargs -I {} echo
    "{}%" | head -n 1
```

which executes, so the step is repaired; at the task level this task scores reward 1.0 (loose pass).

The Baseline $r = 8$ arm starts from the same rejected first attempt: this step obtains an executable command only at its 6th attempt (the first five being rejected), and at the task level the task scores reward 0.67 (loose fail).

D.2 BFCL V4 SINGLE-TURN: FROM A CALL WITH AN EXTRA ARGUMENT TO AN ACCEPTED CALL

Task `bfcl_v4:parallel:parallel_95`, which asks for a separate call of `calculate_magnetic_field` for each of two circular coils. All 8 candidates of Baseline $r = 8$ fail the checker; among them, the one selected is also the outcome of the first attempt (under the continuation rule of §5.1 this paper does not select among attempts), namely

```
[{"function":{"arguments":{"current":10,"permeability":1,"radius":0.5},"name":"
    calculate_magnetic_field"},"type":"function"},
 {"function":{"arguments":{"current":15,"permeability":1,"radius":1},"name":"
    calculate_magnetic_field"},"type":"function"}]
```

From the same first-round start, the TALE arm gives at its 2nd attempt

```
[{"function":{"arguments":{"current":10,"radius":0.5},"name":"calculate_magnetic_field"},"type
    ":"function"},
 {"function":{"arguments":{"current":15,"radius":1},"name":"calculate_magnetic_field"},"type
    ":"function"}]
```

which passes the official checker (the task scores reward 1.0): the function and the argument values are unchanged, and only the extra argument that made the checker fail is gone.

D.3 BFCL V4 MULTI-TURN: FROM A FABRICATED FUNCTION NAME TO A VALID TOOL-CALL BATCH

Task `multi_turn_miss_func_23`. The 1st user turn asks to write some content into `Project_Guide_1.md`, and the `echo` needed to complete the write is deliberately withheld in this turn: the model should explain the situation to the user rather than call a function that does not exist. The two arms emit the same valid call in the first round of this turn (`touch(file_name='Project_Guide_1.md')`); in the round immediately after, their first attempts are also the same,

```
open(file_name='Project_Guide_1.md')
```

and `open` is not in the function table available for this turn, so the batch is rejected (unknown function).

After 1 update in this round, the TALE arm's 2nd attempt no longer repeats `open` but gives a valid `touch(file_name='Project_Guide_1.md')`, and the round ends; in the next round of the same user turn it first tries `open` once more, and after a second rejection turns to natural language, explaining to the user that the file already exists and asking how to proceed—exactly the reaction expected when the needed function is withheld. The task takes 11 model calls and 2 updates in total.

The Baseline $r = 8$ arm starts from the same rejected first attempt: the 8 attempts of this round all give the same call `open(file_name='Project_Guide_1.md')` and the budget is exhausted (the round is recorded as unresolved). The task takes 15 model calls in total, 4 more than TALE.

Both arms pass the scoring of this task—because the needed function is withheld, the scoring does not require the write to actually happen. What this case shows is the behavioral difference within the round: the update moves generation away from the rejected call, whereas retrying without an update repeats it verbatim, so the budget consumption of the round differs (2 attempts against 8).

D.4 SEARCH-R1: FROM SEVERAL ACTIONS IN ONE OUTPUT TO A SINGLE VALID ACTION

Task `nq_test_3349` (Qwen3-1.7B, seed 0). The first attempt shared by the two arms writes two searches into the same output and also generates a retrieval-result block and an answer:

```
<search>how many albums has kelly sold</search>
<search>how many albums has carrie sold</search>
```

It is parsed as 2 actions and cannot be executed as a single action, so the attempt is rejected.

After 2 updates at this step, the TALE arm's 3rd attempt gives a single valid search,

```
<search>best selling albums by Kelly and Carrie</search>
```

and once the retrieval returns, its 4th call gives `<answer>Kelly has sold more albums than Carrie.</answer>`, which SubEM judges correct; the task takes 4 model calls and 2 updates in total.

The Baseline $r = 8$ arm starts from the same rejected first attempt: all 8 attempts are rejected in the same shape (each output carrying 2–3 action tags, 5 of them also generating the retrieval-result block), no retrieval happens, the budget is exhausted, and the task is unresolved.

E Extended Ablations and Comparisons

This appendix gives three groups of extended ablations—E.1 makes the retry budget the independent variable, reconstructing each prefix on the same per-task candidate records and extending the budget to 24; E.2 sweeps the suppression target $\delta_i(\beta)$ and records the pathological serving under small β; E.3 puts the rejected output back into the retry observation (error-as-observation, the mainstream practice of appending a rejected output to the next observation (Shinn et al., 2023)) and measures it on the same task set—and one comparison group, E.4, against the two weight-changing arms.

E.1 Retry Budget: Prefix Reconstruction

This section uses the 692-task subset of BFCL v4 single-turn (the tasks among the official 2,351 that Qwen3-4B got wrong in at least one of three seeds, the same subset as the comparison rows of Appendix E.4). The procedure truncates each arm's own per-task candidate records to prefixes: for each task only the first k candidates are kept ($k = 1, 2, 4, 8$), and the pass count, the call count and the recovery count are recomputed from them. The Baseline $r = 8$ rows come from a source under the same protocol as the main line (historical runs plus a re-run of 24 tasks replacing them under the protocol that shares the first attempt); the three arms coincide at $k = 1$ (22 tasks), because the first attempt is shared by the protocol (§5.2). Table 7 reports the reconstruction.

Table 7: **Prefix reconstruction over the retry budget on the BFCL v4 single-turn 692-task subset (Qwen3-4B).** "Extra calls vs $k = 1$" is the total call count minus 692; "unresolved tasks" counts tasks with no passing candidate when the budget is exhausted; "recovered" counts the tasks recovered after the first rejection.

Arm	k	N	Accuracy	Calls	Extra	Unres.	Recovered
Baseline $r = 8$	1	692	3.2% (22)	692	0	670	0
Baseline $r = 8$	2	692	5.2% (36)	1,362	670	656	14
Baseline $r = 8$	4	692	7.1% (49)	2,665	1,973	643	27
Baseline $r = 8$	8	692	7.8% (54)	5,230	4,538	638	32
TALE $\beta = 1/16$	1	692	3.2% (22)	692	0	670	0
TALE $\beta = 1/16$	2	692	12.0% (83)	1,362	670	609	61
TALE $\beta = 1/16$	4	692	15.3% (106)	2,560	1,868	586	84
TALE $\beta = 1/16$	8	692	16.3% (113)	4,882	4,190	579	91
TALE $\beta = 1/8$	1	692	3.2% (22)	692	0	670	0
TALE $\beta = 1/8$	2	692	11.3% (78)	1,362	670	614	56
TALE $\beta = 1/8$	4	692	14.6% (101)	2,573	1,881	591	79
TALE $\beta = 1/8$	8	692	14.9% (103)	4,935	4,243	589	81

The step-level readings of the main table follow each benchmark's own conventions (Appendix B.2) and differ from this task-level prefix convention.

Three observations. First, retrying itself improves end-to-end task success where the benchmark leaves room for it: the $r = 8$ curve is above $r = 1$ on IC-Bash and BFCL v4 single-turn at every scale, above it on Search-R1 at two of the three scales and level at the third, and level on BFCL v4 multi-turn, with the full per-seed values in Table 11. TALE preserves this retry benefit within the same attempt budget and adds further recovery gains where the local edit has room to act. Second,

the TALE curve rises sharply at the second candidate ($\beta = 1/16$ reaches 83 tasks at $k = 2$, 61 more than at $k = 1$, whereas Baseline gains only 14) and then exhibits diminishing returns as the budget grows (106 at $k = 4$, 113 at $k = 8$, with the calls rising from 2,560 to 4,882), while the update-free curve gains more slowly; the complete prefix reconstruction is in Table 7. Third, the whole curve of $\beta = 1/8$ lies below that of $\beta = 1/16$, consistent with the monotone trend of Appendix E.2.

Extending the budget. Joining the rollouts of the three seeds into a single sequence of 24 attempts (the 8 attempts of seed 0 followed by those of seed 10 and then seed 20; the Baseline $r = 8$ of all three seeds comes from the corrected source under the same protocol as the main line, and the $k \leq 8$ part is the seed-0 curve of the table above), the pass count of the Baseline rises from 54 at $k = 8$ to only

$k = 12$	$k = 16$	$k = 24$
7.9% (55/692)	7.9% (55/692)	8.2% (57/692)

The three seeds reach 54, 51 and 53 tasks with their own 8-attempt budgets; accumulated, the total gain is only 3 tasks (57 at $k = 24$ against 54 at $k = 8$), spread over $k = 9$, $k = 17$ and $k = 20$, and nothing rises after $k > 20$. This 24-attempt curve is still below what TALE reaches at $k = 2$ (83 tasks): continuing to add update-free attempts at the same state brings very limited return, consistent with the dilemma of §2 and the readings of §5.2; raising coverage is itself the existing reading of repeated sampling (Brown et al., 2024), but within the budget this setting can afford, adding calls to update-free attempts did not buy a corresponding number of passes.

E.2 Sweep of the Suppression Target (β)

On the same 692-task subset of BFCL v4 single-turn, with every other setting fixed (the evaluation protocol of §5.1 and the solver parameters of Appendix B.3), we sweep only $\beta \in \{1, 1/4, 1/8, 1/16, 1/24, 1/32\}$, corresponding to a suppression target $\delta_i(\beta) \approx 1/\beta$ from 1 to 32. Each row is an independent run over the 692 tasks; the $\beta = 1/16$ row is the main-line run, the rows with $\beta \in \{1, 1/4, 1/24, 1/32\}$ come from a sweep of the same family, and the $\beta = 1/8$ row comes from another run of the same implementation pipeline. The time column is the end-to-end duration of that run on this subset, so it is not the full-benchmark figure of Table 2. Table 8 lists the sweep.

Table 8: **Sweep of the suppression target on the BFCL v4 single-turn 692-task subset (Qwen3-4B).** Each row is an independent run.

β	$\delta = 1/\beta$	**Accuracy**	**Total calls**	**End-to-end time**
1	1	10.3% (71/692)	5,120	3.83 h
1/4	4	13.7% (95/692)	4,988	4.05 h
1/8	8	14.9% (103/692)	4,935	4.23 h
1/16	16	16.3% (113/692)	4,882	4.53 h
1/24	24	18.2% (126/692)	4,851	7.39 h
1/32	32	18.6% (129/692)	4,836	26.22 h

The pass count ($71 \to 95 \to 103 \to 113 \to 126 \to 129$) rises monotonically with δ and the call count falls with it (5,120 $\to$ 4,836), with no reversal within the swept range. The end-to-end time, however, rises steeply once $\beta \leq 1/24$: 4.53 h at $\beta = 1/16$, 7.39 h at $\beta = 1/24$ and 26.22 h at $\beta = 1/32$.

Pathological serving under small β. The rise in time comes from the generation side: a stronger suppression degenerates the edited distribution on some steps, producing excessively long generations and repetition loops. Table 9 counts the candidate-side behavior of each arm (each arm has about 4,836–5,120 candidates in total).

For $\beta \geq 1/16$ the degenerate candidates are single-digit, while from $\beta \leq 1/24$ they appear at scale. The degeneration takes three forms, one example each (non-answer degeneration is a known phenomenon of autoregressive generation (Holtzman et al., 2020)):

Table 9: **Degenerate candidates under small β (BFCL v4 single-turn 692-task subset, Qwen3-4B).**

β	Mean gen.	p99 gen.	>60 s	>5k chars	Tasks	Length cap
1	1.07 s	3.1 s	0	0	0	0
1/4	1.17 s	3.0 s	2	2	1	2
1/8	1.19 s	3.3 s	0	0	0	0
1/16	1.41 s	4.5 s	2	2	2	2
1/24	3.07 s	4.2 s	30	30	9	30
1/32	14.07 s	352.1 s	180	183	77	180

- **Repetition loops.** The 7th attempt of `bfcl_v4:parallel:parallel_181` (81,920 characters, 340.2 s) is the same token sequence repeated throughout (the trigger condition of such loops, the cost of suppressing them and their resistance to sampling adjustments are matters still being quantified in recent work (Weidmann et al., 2026; Lazaridis et al., 2026; Duan et al., 2026)):

```
</think>

</think>

</think>
...
```

- **Stacked arguments.** The 3rd attempt of `bfcl_v4:live_simple:live_simple_39-16-0` (83,371 characters, 335.4 s) stacks the same group of argument names inside `arguments` over and over (the group appears about 1,800 times):

```
weather_code,weather_description,weather_icon,weather_code,weather_description,
    weather_icon,...
```

- **Degeneration into prose.** The 4th attempt of `bfcl_v4:live_multiple:live_multiple_57-22-4` (83,485 characters, 350.3 s) no longer emits a tool call but unfolds into prose:

```
Okay, let's break down the query to understand what the user is asking for. The user
    mentioned that they received an order for a pair of pants in size UK 34, ...
```

The same shape is visible at $\beta = 1/24$: the 6th, 7th and 8th attempts of `bfcl_v4:multiple:multiple_53` (71,388 characters each, about 333 s) repeatedly stack the same field at `"independent_var": ["interest rates", "unemployment rates"]`.

The main line takes $\beta = 1/16$ in order to stay out of this region: at $\beta = 1/16$ the degenerate candidates are still single-digit (2), whereas from $\beta \leq 1/24$ they appear at scale and directly raise the end-to-end time, and at $\beta = 1/32$ 77 tasks have at least one excessively long candidate and 180 candidates hit the output-length cap of this benchmark (16,384 tokens).

E.3 Putting the Rejected Output Back into the Observation

The retry baseline of the main text (Baseline $r = 8$) uses naive retry: the retry request is the same as the previous one (apart from sampling itself), and the rejected output does not enter the next observation. Another practice writes the earlier failures back into the observation as text, so that the model can observe its own erroneous outputs. We measure this practice on the same task set, to delimit its relation to this paper.

Setup. The task set is the 692-task subset of BFCL v4 single-turn (the same subset as Appendix E.4 and E.1–E.2), with Qwen3-4B, a budget of 8 attempts, every candidate going through the same AST check, and a trigger condition of an invalid tool-call batch or a wrong AST verdict; neither arm performs any weight update (the update count is 0). The *tag* arm uses a fork of the main-line runner: each retry appends a `previous_erroneous_outputs` block after the original observation, containing every rejected output of this task so far (each truncated to 1,200 characters), followed by an instruction:

```
The block above lists your own previous erroneous outputs for this request. Each was rejected
    by the checker. Do not repeat them; produce a corrected tool call that fully satisfies
    the user request.
```

The *off* arm is an independent run of the original runner without that block, the two arms being defined isomorphically to the Baseline $r = 8$ of the main text. Both arms use the same 692-task fixture (the task set is identical task by task). The 58 tasks and 5,214 calls of the off arm here differ by 4 tasks from the Baseline $r = 8$ of Appendix E.1 ($k = 8$, 54 tasks). Table 10 reports the comparison.

Table 10: **Error-as-observation against naive retry (BFCL v4 single-turn 692-task subset, Qwen3-4B, 8 attempts, no weight update).**

Arm	Passed (AST)	Valid	Calls	Used all 8	Pairwise repetition
off arm (naive retry)	8.4% (58/692)	353	5,214	636	0.925
tag arm (error-as-obs.)	9.2% (64/692)	357	5,145	629	0.962

The repetition rate follows the definition of Appendix C.2: over the tasks that used all 8 attempts, the 7 retry candidates are compared pairwise after normalizing the canonical tool call (this section uses Qwen3-4B, unlike the Qwen3-1.7B of Appendix C.2, so the two values are not directly comparable). Paired task by task, the two arms pass 33 tasks together, 31 pass only with the tag arm and 25 only with the off arm, 56 discordant pairs, exact McNemar (McNemar, 1947) $p = 0.50$. Putting the error back into the observation yields no detectable task-level improvement (6 tasks), and neither the call count nor the valid count differs materially (5,145 against 5,214, 357 against 353).

Interpretation. The repetition readings give the mechanistic difference: once the error is written back into the observation, the retry candidates **become more similar to each other** (0.962 against 0.925), while the task-level verdict **remains statistically unchanged** ($p = 0.50$). This is consistent with the mechanism of "changing only the observation, not the distribution": writing the failure back into the context can change what is generated next, while TALE additionally changes the output distribution itself.

Relation to this paper. error-as-observation, and other schemes that change only the observation without changing the weights, are orthogonal to this paper—they act on the observation and sampling side and can be stacked on top of the suppression target of a step. The comparison of this paper still takes naive retry (§5.1), while the observation-side control establishes that this choice is not a strawman: putting the error back yields no detectable task-level gain ($p = 0.50$), and the retry candidates become more similar rather than less. The two mechanisms are therefore complementary: observation-side feedback changes the prompt, whereas TALE changes the local output distribution.

E.4 Comparison with Model-Altering Adaptation Baselines

The two weight-changing controls mentioned in §5.1 are evaluated side by side with the two resampling baselines and TALE under the same problem setting. What they share is that they modify the serving weights after the verifier's judgment: ROSA uses a gradient-based update (weights-only, not intervening on the hidden states) and aTTT a self-updating adapter that persists within an episode. Their scores at this scale are in Table 1 and their wall-clock times in Table 2; this section gives the detailed reading behind those two tables, and the three-seed accuracy behind the single-turn row is in Table 11.

Scope of the comparison. The task sets of the four benchmarks are those of the main table, except that BFCL v4 single-turn uses the 692-task hard subset of Appendix E.1, so that row is reported on the subset's own scale. Table 2 is the exception on the time side: its single-turn column covers the full 2,351-task set, with the tasks outside the hard subset, those that pass at the first attempt, entering at their measured first-attempt cost. That is exact for the arms that work only on a failed step, and a lower bound for ROSA and RA, whose per-task or per-attempt work would continue on those tasks. Within a benchmark the four arms share the task set, the shared first round and the maximum attempt budget; all results in this section are a single run of Qwen3-4B, seed 0. Extra calls are referenced to Baseline $r = 1$ of the same benchmark and the same serving.

Scores. Neither weight-changing control forms a stable advantage over the update-free Baseline $r = 8$. On IC-Bash the unresolved steps are 175 for ROSA and 195 for aTTT, clearly above the 137 of $r = 8$, while TALE is at 86; as a fraction of the triggered steps on each arm's own trajectory, ROSA reaches 74.8% and aTTT 80.6%, against 71.0% for $r = 8$ and 57.3% for TALE. At the task level ROSA scores 60 tasks, aTTT 57 and Baseline $r = 8$ 58, all within the 28.5%–30.0% band and all below the 61 of TALE. On the single-turn subset ROSA scores 62, above the 54 of $r = 8$ (+8 tasks), and aTTT 51, below it (-3), both far below the 113 of TALE. On multi-turn the unresolved ratios of the four rows all sit high (TALE 81.1%, aTTT 90.7%, ROSA 92.6%, $r = 8$ 92.8%) and the task levels are of the same magnitude (ROSA 67, aTTT 56, $r = 8$ 68, TALE 66). On Search-R1 the four rows differ by only 2 tasks (TALE 101, aTTT 101, $r = 8$ 99, ROSA 99); this row has only single-digit triggered steps, so percentages would mislead and the column keeps counts only, with no claim based on the ratios. The local advantages of ROSA appear on IC-Bash (60 against 58) and single-turn (62 against 54), while it is 1 task below $r = 8$ on multi-turn and level with it on Search-R1; aTTT is 2 tasks above $r = 8$ only on Search-R1 and below it on the other three; both are below TALE on IC-Bash and single-turn, on Search-R1 the four rows differ by no more than 2 tasks and constitute no difference, and multi-turn is the one place where the three rows are of the same magnitude as TALE (ROSA 67, aTTT 56, $r = 8$ 68 against TALE 66).

Calls. On calls, Extra calls are referenced to Baseline $r = 1$ of the same serving (the total calls of $r = 1$ are 1,511 / 692 / 5,883 / 677 in order): on IC-Bash TALE is lowest at 668 and $r = 8$ at 1,059, while the two controls rise to 1,315 (ROSA) and 1,447 (aTTT), i.e., on the same failure distribution the weight-changing arms spend 24%–37% more calls than resampling; on single-turn TALE is 4,190, $r = 8$ 4,538, ROSA 4,507 and aTTT 4,541 (aTTT's calls are counted under the same convention as the other three rows, including the injected shared first round); on multi-turn the four rows are nearly level ($r = 8$ 2,712, TALE 2,716, aTTT 2,828, ROSA 2,862); on Search-R1 the total call counts are small by themselves, and the Extra calls of the four rows are TALE 4, aTTT 4, $r = 8$ 10 and ROSA 24. The call structures differ in the same way as the scores: aTTT's per-update cost is lower, but it updates on every failed attempt, so its call total is of the same magnitude as $r = 8$ and ROSA (4,541 against 4,538 and 4,507 on single-turn, and 1,447 above the 1,059 and 1,315 on IC-Bash). ROSA's own cost structure is visible in the run behind its single-turn row: over the tasks where the first attempt fails, its per-task mean is 97.11 s and its median 101.85 s, so its overhead comes from performing a gradient update on every task rather than from a few long tails.

Neither weight-changing control converts the same budget into a stable improvement at the step or the task level: at the step level both are above $r = 8$ on IC-Bash (74.8% / 80.6% against 71.0%) and all three sit above 90% on multi-turn (92.6% / 90.7% against 92.8%); at the task level they exceed $r = 8$ by 2–8 tasks only on individual benchmarks (ROSA on IC-Bash and single-turn, aTTT on Search-R1) and are no higher elsewhere, while the end-to-end time is raised to 1.3–10.0$\times$ that of $r = 8$ (ROSA 2.6–10.0$\times$, aTTT 1.3–2.3$\times$; Table 2). By contrast, §5.2 and Appendix C show that TALE spends fewer calls within the same budget and is no lower than $r = 8$ at the task level on most benchmarks. The gain of TALE therefore comes from turning a rejection that has already happened into a cheap, exactly computable suppression target (§4.1), and an expensive weight update is not necessarily better in score.

E.5 Per-Seed Counts and the Single-Attempt Baseline

Table 11 collects, for every benchmark and model, the per-seed counts of the step-level column (seeds 0 / 10 / 20 in that order) and the task accuracy of the three arms, including **Baseline** $r = 1$, which keeps only the shared first attempt and therefore has no triggered count and zero extra calls (its unresolved rate is 100% by construction, as in Table 1). On Search-R1 with Qwen3-4B the per-seed total calls behind the negative change of TALE's extra-calls entry in the main table are 681 / 662 / 672 against 677 / 663 / 680 for Baseline $r = 1$ (per-seed differences $+4$ / -1 / -8). Baseline $r = 1$ has no triggered column, since nothing triggers a retry under it, but its step-level unresolved count is well defined on its own trajectory: with a single attempt every rejected step stays unresolved, and on IC-Bash these counts are 1,176 for Qwen3-4B and 1,541 for Qwen3-1.7B (three-seed merged totals; Table 11 marks this column as not applicable to the single-attempt baseline).

Table 12 collects the three-seed readings at the three model scales, i.e., the model-scale ablation: the means $\pm$ population standard deviation over seeds 0, 10 and 20, with the changes over the matched

Table 11: **Per-seed counts of the step-level column and task accuracy of the three arms.** Seeds are given in the order 0 / 10 / 20; "—" marks a cell that does not apply: BFCL v4 single-turn has no step column because a task consists of a single step, and Baseline $r = 1$ has no triggered count, since nothing triggers a retry under it. Baseline $r = 1$ keeps only the shared first attempt, hence zero extra calls, and its unresolved rate is 100% by construction, as in Table 1.

Benchmark (task-instances)	Model	Arm	Unresolved / Triggered (s0 / s10 / s20)	Accuracy (%)
IC-Bash (600)	Qwen3-4B	TALE $\beta = 1/16$	86/150, 123/188, 103/182	**31.8 ± 0.9**
		Baseline $r = 8$	137/193, 173/221, 173/226	31.3 ± 1.7
		Baseline $r = 1$	—	29.0 ± 1.1
BFCL v4 single (7,053)	Qwen3-4B	TALE $\beta = 1/16$	—	**75.5 ± 0.1**
		Baseline $r = 8$	—	72.8 ± 0.05
		Baseline $r = 1$	—	71.5 ± 0.03
BFCL v4 multi (1,872)	Qwen3-4B	TALE $\beta = 1/16$	330/407, 331/398, 329/416	11.2 ± 0.5
		Baseline $r = 8$	372/401, 371/399, 380/417	**11.4 ± 0.3**
		Baseline $r = 1$	—	**11.4 ± 0.2**
Search-R1 (900)	Qwen3-4B	TALE $\beta = 1/16$	1/3, 0/1, 0/0	**33.7 ± 0.0**
		Baseline $r = 8$	1/2, 0/1, 0/0	32.7 ± 0.7
		Baseline $r = 1$	—	32.8 ± 0.6
IC-Bash (600)	Qwen3-1.7B	TALE $\beta = 1/16$	236/302, 204/287, 226/299	**25.8 ± 0.9**
		Baseline $r = 8$	231/310, 249/325, 242/314	25.7 ± 1.0
		Baseline $r = 1$	—	24.2 ± 1.7
BFCL v4 single (7,053)	Qwen3-1.7B	TALE $\beta = 1/16$	—	**70.5 ± 0.1**
		Baseline $r = 8$	—	66.7 ± 0.1
		Baseline $r = 1$	—	65.4 ± 0.1
BFCL v4 multi (1,872)	Qwen3-1.7B	TALE $\beta = 1/16$	49/75, 50/83, 52/80	3.3 ± 0.2
		Baseline $r = 8$	54/62, 60/79, 60/66	**3.3 ± 0.3**
		Baseline $r = 1$	—	**3.3 ± 0.3**
Search-R1 (900)	Qwen3-1.7B	TALE $\beta = 1/16$	31/121, 26/112, 32/104	**13.1 ± 0.9**
		Baseline $r = 8$	47/115, 49/108, 47/100	12.7 ± 0.3
		Baseline $r = 1$	—	8.8 ± 1.1
IC-Bash (600)	Qwen3-8B	TALE $\beta = 1/16$	154/222, 148/231, 167/233	**31.2 ± 1.0**
		Baseline $r = 8$	233/277, 181/243, 215/269	30.2 ± 0.9
		Baseline $r = 1$	—	28.5 ± 0.7
BFCL v4 single (7,053)	Qwen3-8B	TALE $\beta = 1/16$	—	**77.7 ± 0.0**
		Baseline $r = 8$	—	75.3 ± 0.1
		Baseline $r = 1$	—	74.8 ± 0.05
BFCL v4 multi (1,872)	Qwen3-8B	TALE $\beta = 1/16$	78/104, 74/93, 74/99	8.8 ± 0.2
		Baseline $r = 8$	86/95, 84/91, 87/92	8.7 ± 0.2
		Baseline $r = 1$	—	**8.8 ± 0.3**
Search-R1 (900)	Qwen3-8B	TALE $\beta = 1/16$	3/9, 2/10, 2/11	32.4 ± 0.6
		Baseline $r = 8$	2/8, 3/10, 3/10	**33.1 ± 0.6**
		Baseline $r = 1$	—	32.4 ± 0.6

Baseline $r = 8$. It is the multi-seed and multi-scale counterpart of the seed-0 comparison of Table 1, and its ratio entries are the per-seed ratios averaged over the three seeds.

F Related Work

This appendix organizes prior work along two recovery lines and three neighboring lines, giving at the end of each paragraph its difference from or relation to this paper.

Recovery that does not change the weights. This line contains three practices: putting the error or the verifier feedback back into the context and continuing (error-as-observation, the default of mainstream harnesses and the ReAct-style loop), resampling at the same state or drawing more candidates, and selecting among several candidates (best-of-n, self-consistency and verifier-guided selection, trading more sampling for higher coverage) (Yao et al., 2023; Wang et al., 2023; Snell et al., 2025). The cheapest modification in this line is deduplicated resampling, which does not change the model and only excludes the already observed strings at sampling time; the mecha-

Table 12: **Three-seed readings at three model scales.** Mean ± population standard deviation over seeds 0, 10 and 20; the second entry in each cell is the change over the matched Baseline $r = 8$, in percent (accuracy: percentage points) and shown in red where it is an improvement, with bold marking the better value. Where a mean is itself negative, the cell keeps the count only, since a percentage over that reference is not stable (the Search-R1 extra calls at Qwen3-4B, footnote †). The four benchmarks use 200 / 2,351 / 624 / 300 tasks per seed; Baseline $r = 1$, the per-seed counts and the per-seed calls are in Appendix E.5.

Benchmark	Model	Arm	Unresolved / Triggered	Extra Calls vs. $r = 1$	Accuracy (%)
IC-Bash	Qwen3-4B	TALE $\beta = 1/16$	**59.8% ± 4.0%** (−20.6%)	**805 ± 117** (−33.3%)	**31.8% ± 0.9%** (+0.5 pp)
		Baseline $r = 8$	75.3% ± 3.1%	1,206 ± 105	31.3% ± 1.7%
BFCL v4 single	Qwen3-4B	TALE $\beta = 1/16$	—	**4,184 ± 10** (−8.0%)	**75.5% ± 0.1%** (+2.7 pp)
		Baseline $r = 8$	—	4,549 ± 11	72.8% ± 0.05%
BFCL v4 multi	Qwen3-4B	TALE $\beta = 1/16$	**81.1% ± 1.7%** (−12.1%)	**2,646 ± 74** (−2.5%)	11.2% ± 0.5% (−0.2 pp)
		Baseline $r = 8$	92.3% ± 0.8%	2,713 ± 34	**11.4% ± 0.3%**
Search-R1	Qwen3-4B	TALE $\beta = 1/16$	—‡	−1.7† **± 4.9**	**33.7% ± 0.0%** (+1.0 pp)
		Baseline $r = 8$	—‡	6 ± 3	32.7% ± 0.7%
IC-Bash	Qwen3-1.7B	TALE $\beta = 1/16$	**74.9% ± 2.9%** (−1.5%)	**1,663 ± 74** (−9.4%)	**25.8% ± 0.9%** (+0.2 pp)
		Baseline $r = 8$	76.1% ± 1.1%	1,836 ± 54	25.7% ± 1.0%
BFCL v4 single	Qwen3-1.7B	TALE $\beta = 1/16$	—	**5,050 ± 3** (−9.0%)	**70.5% ± 0.1%** (+3.8 pp)
		Baseline $r = 8$	—	5,549 ± 11	66.7% ± 0.1%
BFCL v4 multi	Qwen3-1.7B	TALE $\beta = 1/16$	**63.5% ± 2.3%** (−25.0%)	**369 ± 73** (−13.4%)	3.3% ± 0.2% (−0.1 pp)
		Baseline $r = 8$	84.7% ± 6.4%	426 ± 49	**3.3% ± 0.3%**
Search-R1	Qwen3-1.7B	TALE $\beta = 1/16$	**26.5% ± 3.2%** (−40.3%)	**467 ± 24** (−11.3%)	**13.1% ± 0.9%** (+0.4 pp)
		Baseline $r = 8$	44.4% ± 2.6%	526 ± 28	12.7% ± 0.3%
IC-Bash	Qwen3-8B	TALE $\beta = 1/16$	**68.4% ± 3.2%** (−14.0%)	**1,179 ± 71** (−24.1%)	**31.2% ± 1.0%** (+1.0 pp)
		Baseline $r = 8$	79.5% ± 3.9%	1,553 ± 150	30.2% ± 0.9%
BFCL v4 single	Qwen3-8B	TALE $\beta = 1/16$	—	**3,783 ± 5** (−7.6%)	**77.7% ± 0.0%** (+2.4 pp)
		Baseline $r = 8$	—	4,093 ± 8	75.3% ± 0.1%
BFCL v4 multi	Qwen3-8B	TALE $\beta = 1/16$	**76.4% ± 2.2%** (−17.3%)	**579 ± 25** (−7.0%)	8.8% ± 0.2% (+0.1 pp)
		Baseline $r = 8$	92.5% ± 1.7%	623 ± 8	8.7% ± 0.2%
Search-R1	Qwen3-8B	TALE $\beta = 1/16$	—‡	41 ± 10 (+53%)	32.4% ± 0.6% (−0.7 pp)
		Baseline $r = 8$	—‡	**27 ± 4**	**33.1% ± 0.6%**

† A Search-R1 task ends at its final answer (early stopping), so the steps it executes, and hence this difference, can vary between arms and be negative (per-seed calls in Table 11). ‡ The Search-R1 rows of Qwen3-4B and Qwen3-8B trigger only single-digit steps, so no rate is reported for them in this column; the per-seed counts of all three scales are in Table 11.

nism comparison below approximates this exclusion with a large negative bias and measures one approximate implementation of it. Going further, self-correction organizes the failure into a natural-language critique or an optimization signal (Reflexion, Self-Refine, CRITIC, TextGrad), adding a layer of linguistic feedback (Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024; Yuksekgonul et al., 2025). Constrained decoding (PICARD, XGrammar, NeuroLogic and the like, structured outputs) moves part of the judgment before generation (Scholak et al., 2021; Dong et al., 2025; Lu et al., 2021). Benchmarks have recently appeared that take error localization and the recovery process as their object (Guan et al., 2026), which is the process this paper intervenes in. Targeted interventions on the harness side have also been reported recently: hanging a pre-written instruction on the state before a failure, or rejecting one tool call, can raise repeated success on the same task without changing capability (Agarwal & Jain, 2026); the policies there are written by hand for a failure family and frozen, whereas the update of this paper is produced automatically by the rejection event and checked item by item on a real recomputation (§3.3). The error-as-observation part of this line is orthogonal to this paper: it changes the observation while this paper changes the distribution of the next output, so the two can be stacked and each can be measured (Appendix E.3). The comparison of this paper deliberately keeps naive retry (§5.1): the retry request merely repeats the original one, and neither the rejected output nor the error message enters the next observation, so the difference between Baseline $r = 8$ and TALE comes only from the update itself; had the observation side been optimized as well, the gain of TALE would be confounded with that of the prompt and the difference between the two arms would no longer be attributable. This choice does not weaken the

comparison: putting the error back into the observation adds only 6 tasks ($p = 0.50$) while raising the mutual similarity of the retry candidates to 0.962 (against 0.925 for naive retry), so resampling still draws from the original failure region (Appendix E.3). Constrained decoding covers only errors expressible in a static grammar and is orthogonal to this paper: it removes what its contract rejects, while the failures that remain are syntactically valid and still wrong (Park et al., 2024; Özgü et al., 2026) (§5.2). **Difference from this paper**: this line changes the distribution of the next output little or not at all, and therefore has difficulty moving generation away from the failed continuation; the update of this paper is triggered by the rejection event and directly lowers the likelihood of the rejected sequence, and the repetition of the retry candidates and the number of attempts needed to recover are the measurable readings of that difference (§5.2, Appendix C.2, Appendix E.1).

Weight-changing test-time adaptation. This class changes the model between attempts, with two subclasses both appearing in the main text: test-time parameter adaptation (directly changing the weights without touching the hidden states) and self-updating adapters carried within an episode. Earlier test-time adaptation and test-time training put the adaptation object on the whole test stream and unfold along two directions, "changing the model state with test-time signals" and "changing the prediction with extra inference compute" (Niu et al., 2026); continuous updating then brings drift and catastrophic forgetting and requires explicit reset mechanisms. Recent work that moves this line onto LLM agents likewise updates continuously within an episode (Acikgoz et al., 2026); on the offline side, preference or single-point feedback writes such signals into the weights and requires a full training pipeline (Rafailov et al., 2023; Ethayarajh et al., 2024), as does self-correction trained into the model rather than prompted (Kumar et al., 2024). ROSA and TTRL adjust the model continuously with test-time interaction or feedback (Wei et al., 2025; Zuo et al., 2025); aTTT extracts its signal from n-grams or behavior patterns that repeat within an episode, and its adapter persists for the whole episode (Wang et al., 2026); the newest branch takes the final projection of an MLP block as adaptable fast weights, so that it can be attached to an existing LLM and updated in place at inference time (Feng et al., 2026), with its update likewise persisting on the stream; such updates are usually realized with low-rank adapters (Hu et al., 2022). The line keeps evolving in later work that jointly optimizes the textual side and the weight update (ROSA2, (Wei et al., 2026)); the comparison of this paper still takes the single-axis weight update of ROSA, to keep the comparison variable single. **Difference from this paper**: their adaptation object is an episode (the adapter of aTTT) or a task (the update of ROSA)—whereas in the multi-step benchmarks of this paper one step corresponds to such a task, so the update is cleared at the end of the step (Appendix B.3); the update of this paper is driven by the rejection event of a single step, serves only the next attempt, and is removed after the next judgment (§3.4). The readings and costs of ROSA and aTTT as representatives of this line under our setting are in Appendix E.4.

The state-changing line. Backtracking, replanning and tree search change the state of the trajectory itself (Hao et al., 2023; Zhou et al., 2024). **Relation to this paper**: orthogonal and composable—this paper asks what to do when an acceptable action must be delivered at this state.

Logit- and representation-level interventions. Contrastive decoding, inference-time intervention, representation engineering (including activation interventions along a fixed direction) and unlikelihood suppress unwanted outputs by changing the relative scores of certain tokens or directions (Li et al., 2023b; Chuang et al., 2024; Li et al., 2023a; Zou et al., 2023; Turner et al., 2023; Welleck et al., 2020). **Difference from this paper**: these interventions are applied per token or per direction and are unaware of which sequence they land on; any other output sharing those tokens is suppressed as well, and the intervention has no way to know which sequence was just rejected. When the correct answer differs from the rejected string by only a few tokens, it is suppressed too. The target of this paper is produced by this rejection, is defined on the log-probability of the rejected sequence itself, and disappears when the step ends (the perturbation it causes to the remaining outputs is given by the KL term of §4.2). The closest special case in this line is the decoding-time logit ban: it lowers the tokens of the most recently rejected continuation at retry time, likewise without changing the weights, and the tokens it suppresses are likewise produced by the string just rejected (not a fixed cross-input bias), but it still applies at the token level, whereas this paper lowers the likelihood of the rejected sequence itself: the suppression is bounded, its achievement is checked item by item (§4.2, Appendix C), and no hard exclusion is applied to any token. The cost and limits of replacing suppression with a token mask have their own readings in the literature: a rigid mask distorts the model's

own distribution and pushes generation toward legal but worse outputs (Park et al., 2024; Özgü et al., 2026), hard constraints at the format level substantially weaken reasoning (Tam et al., 2024); once the suppression is strong enough to hold down a loop, it also damages formatting and fluency (Weidmann et al., 2026), while loops themselves survive most sampling adjustments (Lazaridis et al., 2026); writing the target onto individual token probabilities also yields only marginal gains on the training side (Su et al., 2022; Jiang et al., 2022). We measured this mechanism comparison on the 692-task subset of BFCL v4 single-turn (the same subset as Appendix E.4 and Appendix E.1): three arms—an anchor without a ban, a weak arm suppressing the first token (bias -20, at most 1 token) and a strong arm suppressing the whole token set (bias -80, at most 256 tokens)—under the same 8-attempt budget and AST check give correct counts of 372 $\rightarrow$ 366 $\rightarrow$ 366 and valid counts of 676 $\rightarrow$ 677 $\rightarrow$ 678, i.e., **no gain at this scale**; under the strong arm another 162 candidates are truncated after reaching the 16,384-token output cap (with finish reason `length`), so a hard exclusion **pushes generation toward degenerate long outputs**. A paired example (`bfcl_v4:live_multiple:live_multiple_274-127-9`) shows this degenerate shape: the first attempt of both arms is the same, a `Weather_1_GetWeather(city="Chcicago", date="2024-02-13")` with the city name misspelled (a legal tool call that the AST judges wrong, hence triggering the next attempt); the arm without a ban gives a candidate of the same shape at its next attempt, while the ban arm (suppressing the 39 tokens of that string) degenerates in the same attempt into nested repetition until the 16,384-token cap truncates it, 458,936 characters in total:

```
<tool_response>

{
  "__type__" : "",
  "__function__" : {
    "__type__" : "",
    "__function__" : {
      "__type__" : "",
      ...
```

Both arms then correct the city name at their respective next attempts and complete the task: the ban brought no earlier recovery, it only replaced an ordinary candidate with a degenerate output that reached the output cap. This comparison group was run on a different serving configuration from the main line (speculative decoding disabled), so its absolute readings are not compared with the main table; this section takes only its internal contrast: merely excluding an observed string from sampling is not enough to buy recovery—consistent with the readings of Appendix C.2, the update changes the probability structure around the rejected sequence rather than excluding that one string.

Closed-form model editing. Another neighboring family writes new facts into the weights with a constrained least squares (Meng et al., 2022), whose family and taxonomy are surveyed recently (Salehoof et al., 2025). They share with this paper the closed-form, gradient-free nature, but are opposite in three places: the direction (injecting facts vs suppressing failures), the location (deep MLPs vs the output side) and the lifecycle (permanent vs a single step): the suppression target of this paper is defined by a single rejection and is cleared after the next verifier judgment (§3.4).