PRICE: LEARNING WHAT TO RECOMPUTE FOR REASONING IN DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models can revise outputs, but choosing what to recompute is difficult. Local repair may leave inconsistent downstream results. Recomputing all descendants may waste useful work. We propose PRICE (Predictive Repair via Intervention-trained Candidate Evaluation). It chooses semantic repair scopes using learned estimates of final reward and remaining cost. A predicted dependency graph proposes a bounded pool of executable actions. Offline continuations from the same partial state train estimates of reward and cost differences. Inference runs one action without trying alternative continuations. The executor hides predicted affected text from the model. Cached values return only after a check without reference answers; its cost is counted. Our analysis shows when selective repair beats full closure and when learned selection preserves this gain. It bounds regret from missing candidates and value errors. Across three instruction-tuned diffusion backbones and four reasoning benchmarks, evaluated settings improve scores over full closure with fewer model evaluations. Gains depend on candidate coverage, value accuracy, and the reliability and full cost of checked reuse.

1 INTRODUCTION

Discrete diffusion offers an alternative to autoregressive generation (Austin et al., 2021a; Lou et al., 2024). Masked diffusion language models (dLLMs) denoise partially visible sequences, updating multiple positions together (Sahoo et al., 2024). LLaDA, iLLaDA, and Dream scale this interface (Nie et al., 2025; 2026; Ye et al., 2025). Remasking makes generated content revisable (Huang et al., 2026; Yao, 2026). However, correcting an intermediate result can make downstream work invalid. This raises a question: **When a reasoning result changes, which existing reasoning units should be recomputed together?**

Recent work explores remasking and refinement (Huang et al., 2026; Zhang et al., 2025; Park et al., 2026; Ren et al., 2026), learned generation order and structural guidance (Jazbec et al., 2026; Kim et al., 2026; Israel et al., 2026), context control (Fomenko et al., 2026), and value- or search-guided diffusion inference (Jeon et al., 2026; Misaki & Akiba, 2026). These mechanisms identify unreliable content, structure generation, and revisit partial states. The remaining decision is which complete repair scope is worth executing, given its downstream benefit, remaining cost, and reuse opportunities.

For example, let x change from 4 to 5, with descendants $y = 2x$ and $z = \mathbf{1}[x > 0]$, where $\mathbf{1}$ is the indicator function. The correct y changes from 8 to 10, while z remains 1. Updating only x leaves stale y. Regenerating both descendants discards a valid result. A dependency graph identifies results that may be affected. The value of repair also depends on current content, remaining budget, and checking cost.

PRICE learns the reward–cost value of complete repair actions at the current state to guide repair-scope selection. The number of possible scopes can be large. PRICE uses a predicted dependency graph to construct a bounded set of candidate scopes. Offline interventions share the same starting state, continuation policy, executor, and budget. Their outcomes provide training targets for reward gains and changes in full remaining cost, both relative to no repair. At inference, PRICE uses pre-action state features to select one scope for recomputation and checked reuse.

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.

In a bidirectional model, visible stale descendants can influence upstream replacement. PRICE quarantines the predicted affected region by hiding its text from the model. It discards selected units and caches the rest outside the model. Cached content returns only after its current parents are ready and a check accepts it. The check uses no reference answer, and its cost is counted. Content may therefore be eligible for reuse without being visible. Action costs include failed checks and the regeneration that follows.

Our theory shows when computation savings outweigh reward loss from incorrect reuse. Learned selection preserves this gain when it exceeds twice the value error plus the deployment margin. Total regret separates missing candidates from errors in selecting among them.

Across LLaDA-8B-Instruct, iLLaDA-8B-Instruct, and Dream-v0-Instruct-7B on GSM8K, MATH, HumanEval, and MBPP, PRICE improves task scores over full-closure repair by 0.9–2.5 percentage points while reducing four-task macro-averaged model evaluations by 12.0–12.3%. On sampled LLaDA repair states, the estimated best-in-pool action differs from full closure in 79.0% of states, averaged equally across the four tasks. Full closure is thus often not the estimated best scope in the measured pool. Scorer and execution ablations examine how action ranking and reuse rules contribute to these gains. Our contributions are:

1. **Learning what to recompute.** We learn the value of complete repair scopes at each state. These values guide action selection to optimize final reward and remaining computation cost.
2. **An executable repair framework.** We develop matched intervention learning and an executor for selective retention. It combines scope selection, quarantine, regeneration, and verification-gated reuse, running one action at a time.
3. **The value and limits of selective repair.** We establish when selective repair improves reward–cost utility over full closure and when learned selection preserves this gain. We bound regret using candidate coverage and value accuracy. Experiments across three backbones and four benchmarks measure quality–cost gains and isolate the roles of scope selection, scoring, and execution.

2 Related Work

Diffusion of Thought studies denoising-based reasoning and self-correction (Ye et al., 2024). Remasking diffusion models (ReMDM) add a remasking backward process (Wang et al., 2025); the Remasking-enabled Diffusion Language Model (RemeDi) learns token-quality signals (Huang et al., 2026). Corrective Diffusion Language Models supervise erroneous visible tokens (Zhang et al., 2025). D3IM revises tokens directly, while Token-to-Mask (T2M) reopens them for denoising (Yu et al., 2026; Yao, 2026). Post-hoc Unlocking and REfinement (PURE) refines unstable draft windows (Park et al., 2026); edit-based ME-DLM learns replacement, deletion, and insertion (Ren et al., 2026). We study which dependent reasoning units to recompute together, accounting for downstream reward and the full cost of repair and reuse.

Block Diffusion orders blocks autoregressively (Arriola et al., 2025); Set Diffusion orders token sets flexibly (Arriola & Kuleshov, 2026). Fast-dLLM combines confidence-based parallel decoding and approximate KV caching (Wu et al., 2026a). The Entropy Bounded Sampler (EB-Sampler) bounds dependence among parallel updates (Ben-Hamu et al., 2025). Confidence-Oriented Residual Acceptance for Diffusion Language Models (CORA-Diff) accepts residual tokens based on confidence and persistence. It preserves transfer rules and accepted context (Wu et al., 2026b). Learned policies optimize unmasking order (Jazbec et al., 2026). d1 combines masked SFT and diffu-GRPO, a diffusion variant of GRPO (Zhao et al., 2025). DiffuCoder uses coupled-GRPO for code (Gong et al., 2026). Dependency-Aware Parallel Decoding (DAPD) uses attention-derived dependencies (Kim et al., 2026); Planned Diffusion plans independent chunks for parallel denoising (Israel et al., 2026). Reward-free guidance (RFG) uses policy/reference likelihood ratios (Chen et al., 2025); Masked-Diffusion-Model Value-Guided Backtracking (MDM-VGB) guides unmasking and remasking with partial-state values (Jeon et al., 2026). UnMaskFork searches unmasking actions via MCTS (Misaki & Akiba, 2026); rational metareasoning balances answer utility and token cost (De Sabbata et al., 2024). PRICE selects the repair scope; a fixed advancement rule derives the eligible updates from the cleared state.

Self-Refine revises using self-feedback (Madaan et al., 2023); Reflexion reuses reflections across trials (Shinn et al., 2023). Diffusion-styled chain-of-thought reasoning (DiffCoT) refines windows under autoregressive decoding (Cao et al., 2026); CausalFlow validates minimal repairs through interventions and downstream re-execution (Bonagiri et al., 2026). NAVIRA masks suspect tokens before computing replacement logits (Fomenko et al., 2026). Detect–Remask–Repair selectively regenerates outdated summary spans (Zou et al., 2026). Reward-guided stitching selects steps across diffusion trajectories with a process reward model, conditioning an autoregressive solver on the combined rationale (Miles et al., 2026). PRICE learns the downstream reward–cost values of complete semantic repair actions from matched offline continuations. It selects scopes without online branching, hides affected results, and restores retained values only after verification.

3 METHOD

PRICE selects repair scope R and derives eligible update set U by a fixed rule. It scores complete actions and reuses values that pass checks (Figure 1). Matched offline continuations train relative reward and normalized-cost estimates. Appendix D summarizes the notation.

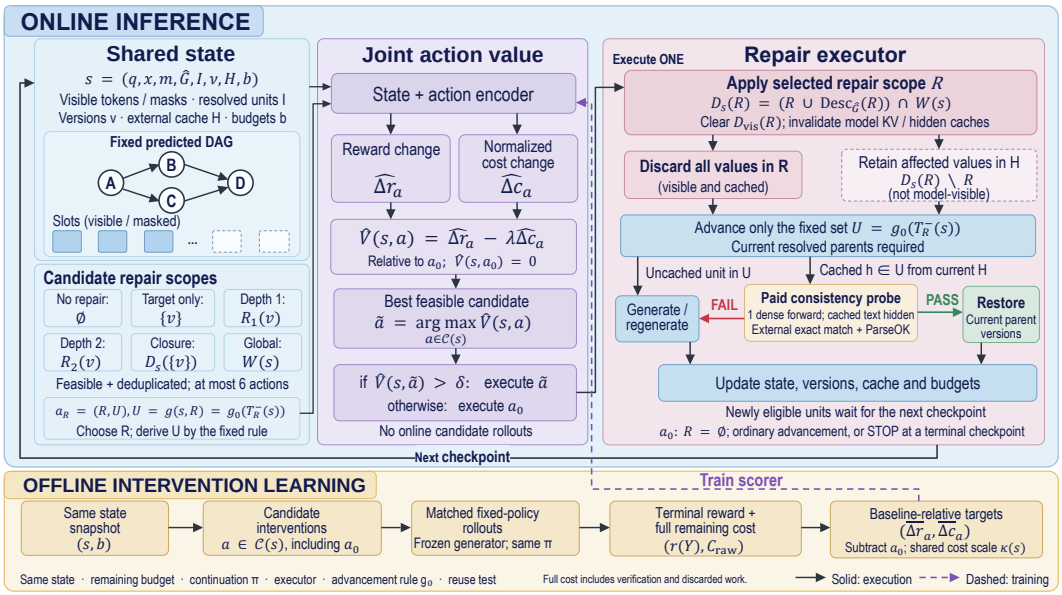


Figure 1: **PRICE: repair selection and verification-gated reuse.** Offline rollouts train estimates of reward and normalized remaining-cost differences relative to the no-repair baseline a_0. Online, the controller selects one action $a_R = (R, g(s, R))$, with $g(s, R)$ the induced update set. The executor clears the visible affected region $D_{\mathrm{vis}}(R)$, discards R, and restores eligible cached values only after a consistency check whose cost is counted. Dashed arrows denote training.

3.1 PROBLEM FORMULATION: REPAIR AS STATE-CONDITIONED ACTION SELECTION

Given only question q, the frozen backbone predicts a DAG $\widehat{G} = (V, E)$ of reasoning units V and dependency edges E (Appendix C.1). Units have typed results and fixed token slots. The graph stays fixed within an episode. Planning and parser failures use a budgeted fallback and remain in evaluation. The partial state is $s = (q, x, m, \widehat{G}, I, \nu, H, b)$. Here x, m specify visible tokens and masks, and I contains resolved units. ν records versions and discard flags, H stores stale text outside the model, and b contains remaining budgets. Resolution requires successful parsing, filled required positions, and current parent versions. It does not imply correctness. Writing $\mathrm{Pa}_{\widehat{G}}(v)$ for the parents of unit v, the eligible frontier is $F_{\widehat{G}}(I) = \{v \in V \setminus I : \mathrm{Pa}_{\widehat{G}}(v) \subseteq I\}$. The predicted graph thus limits eligible updates, even when its edges are wrong.

The repair scope R is the decision variable; advancement follows a fixed shared rule. Let $W_{\mathrm{vis}}(s)$ contain units with generated tokens on the visible canvas, including partially filled slots, and let

$W_{\text{cache}}(s)$ contain units with stored content only in the external cache H. Each unit has one active stored value, so these sets are disjoint, with $W(s) = W_{\text{vis}}(s) \cup W_{\text{cache}}(s)$. For $R \subseteq W(s)$, write $\text{Desc}_{\widehat{G}}(R)$ for graph descendants of R. The affected stored region is

$$D_s(R) = \big(R \cup \text{Desc}_{\widehat{G}}(R)\big) \cap W(s). \tag{1}$$

Its visible and cached parts are $D_{\text{vis}}(R) = D_s(R) \cap W_{\text{vis}}(s)$ and $D_{\text{cache}}(R) = D_s(R) \cap W_{\text{cache}}(s)$. Write $D(R) = D_s(R)$ and let $T_R^-(s)$ be the cleared state after invalidation and cache handling, before generation or verification. With $I^- = I \setminus D(R)$, the executable action is

$$U = g(s, R) := g_0(T_R^-(s)) \subseteq F_{\widehat{G}}(I^-), \qquad a_R = (R, g(s, R)). \tag{2}$$

The deterministic rule g_0 is fixed across training and evaluation; scopes can induce different eligible sets under this shared policy. The feasible space $\mathcal{A}(s)$ contains these induced actions; R remains the decision variable.

Let Y be the final output and C_{raw} the actual remaining cost. For a fixed continuation policy π, remaining budget b, and positive state-level cost scale $\kappa(s)$ shared by all candidates, let $P_b^\pi(s, a) = \mathbb{E}[r(Y) \mid s, a, \pi, b]$ denote expected reward and $\mu_C(s, a) = \mathbb{E}[C_{\text{raw}} \mid s, a, \pi, b]$ expected raw remaining cost. The normalized reward–cost value is

$$Q_{\lambda,b}^\pi(s, a) = P_b^\pi(s, a) - \lambda C_b^\pi(s, a), \qquad C_b^\pi(s, a) = \mu_C(s, a)/\kappa(s), \quad \lambda \geq 0. \tag{3}$$

Here $r(Y) \in [0, 1]$ is final task reward and λ weights cost. We abbreviate $P_b^\pi, C_b^\pi, Q_{\lambda,b}^\pi$ as P, C, Q when π, b, λ are fixed. Let a_0 denote no repair with ordinary advancement, or STOP at a terminal checkpoint. Use $\kappa(s) = \mu_C(s, a_0)$ when positive. A zero-cost baseline needs a separate positive scale shared by all candidates. Raw remaining cost includes all charged operations after this state, including checks and work later discarded. The number of function evaluations (NFE) counts dense backbone calls, including planning. End-to-end time also includes scoring and state/cache work. Earlier work is shared and counted once per episode. Every action starts with the same remaining budget.

3.2 Candidate Repair Scope Construction

Considering every subset can require exponentially many scopes. We instead construct a bounded pool $\mathcal{C}(s) \subseteq \mathcal{A}(s)$ of feasible actions. For a target $v \in W(s)$ selected using current observations, the pool includes no repair ($R = \varnothing$), target-only repair ($R = \{v\}$), depth-bounded scopes $R_1(v)$ and $R_2(v)$, full predicted closure ($R = D(\{v\})$), and global discard ($R = W(s)$). Each $R_k(v)$ contains v and stored descendants within shortest directed path distance k; these scopes need not be ancestor-closed. Global discard ignores the graph when choosing stored content to remove; generation still follows the executor's rules for eligible updates. Ranking, tie-breaking, and feasibility rules are fixed before collection; filtering and removing duplicates leave at most six deployment actions.

Equation 2 determines advancement for each scope. The pool always includes a_0. Appendices C.1–C.3 specify planning, deterministic depth-based construction, shared controls, and online information.

3.3 Joint Action Value Learning

Combined repairs can help each other, overlap in their effects, or cause harm. For disjoint regions A, B whose four associated actions are feasible, define reward, cost, and utility interactions under the same advancement rule, executor, continuation, budget, and state normalization:

$$\mathcal{I}_s^Z(A, B) = Z(s, a_{A\cup B}) - Z(s, a_A) - Z(s, a_B) + Z(s, a_\varnothing), \quad Z \in \{P, C, Q\}. \tag{4}$$

Since $\mathcal{I}_s^Q = \mathcal{I}_s^P - \lambda \mathcal{I}_s^C$, both reward interactions and shared computation affect repair value. The encoder represents the entire a_R, including induced advancement, and compares its value with no repair and other candidates.

The encoder pools available unit representations, confidence and entropy statistics, mask fractions, graph relations, version history, budgets, and membership flags for R and U. With reference action a_0, its two heads predict $\Delta r_a = P(s, a) - P(s, a_0)$ and $\Delta c_a = C(s, a) - C(s, a_0)$. The relative action value is

$$V(s, a) = Q(s, a) - Q(s, a_0), \qquad \widehat{V}(s, a) = \widehat{\Delta r}_a - \lambda \widehat{\Delta c}_a. \tag{5}$$

We set $\widehat{V}(s, a_0) = 0$. Only pre-action observations are available online; extra probes are shared and charged.

3.4 Offline Intervention Learning

With the generator frozen, collect partial states under a fixed policy π. Clone each state, execute candidate interventions, and continue under the same policy and remaining budget. Repeated rollouts estimate differences from a_0, normalizing costs by the shared state-level scale $\kappa(s)$. Interaction diagnostics collect $\varnothing, A, B, A \cup B$ with a shared executor, advancement rule, and reuse test. Corresponding random events are paired by problem, slot, position, and local update count. Appendix C.5 details event pairing, problem-level splits, and policy refresh.

Writing the estimated targets as $\overline{\Delta r}_a$ and $\overline{\Delta c}_a$, train with

$$\mathcal{L} = \mathbb{E}_{s,a}\big[(\widehat{\Delta r}_a - \overline{\Delta r}_a)^2 + \beta\,\mathrm{Huber}\big((\widehat{\Delta c}_a - \overline{\Delta c}_a)/c_0\big)\big], \tag{6}$$

where $\beta \geq 0$ weights the cost loss and $c_0 > 0$ scales its residual during training. The expectation averages training state–action pairs. Reference answers score outcomes but do not supply replacement values. Failed continuations keep their outcome and cost.

3.5 Selective Retention Execution

Executing a_R separates which values are discarded from which are visible. Only $D_{\mathrm{vis}}(R)$ is removed from the visible canvas: generated token identifiers are replaced by masks while slot positions remain allocated. Values already in $D_{\mathrm{cache}}(R)$ stay outside model inputs. Model KV and hidden-state caches based on removed text are invalidated. Dense attention may require rebuilding them fully. These caches differ from the external text store H.

All active values in R are discarded, including any external cached copies. Visible values in $D(R) \setminus R$ move into H; already cached values there remain hidden. Their earlier checks are no longer valid. Source-version metadata is kept for a new check against current parents. Changing a version tag alone never permits reuse. This defines $T_R^-(s)$. Within the control interval, only units in the fixed set $U = g(s, R)$ advance: uncached units generate or regenerate, while cached units receive the probe below. Newly eligible units wait for the next interval.

One-pass consistency probe. For a cached unit $h \in U$, let c_h denote its cached token sequence and J_h its allocated slot. Once its parents have current resolved versions, mask the entire slot and perform one dense forward without exposing c_h. Let x^- denote this cleared input and p_θ the token distribution of the frozen generator with parameters θ. A fixed shared decoding rule d_{check} produces a probe sequence:

$$\widetilde{c}_h = d_{\mathrm{check}}\big(\{p_\theta(x_j = \cdot \mid x^-)\}_{j \in J_h}\big), \qquad \mathrm{Verify}(h) = \mathbf{1}[\widetilde{c}_h = c_h \ \wedge\ \mathrm{ParseOK}(\widetilde{c}_h)]. \tag{7}$$

Here ParseOK checks the required output format. The external comparator reads the cache only after prediction. An accepted value is restored with current parent versions. Rejection starts ordinary regeneration. Each check costs one NFE, and all later forwards are counted. Appendix C.4 specifies exact-match serialization, decoding, and rejection continuation. The check tests consistency rather than semantic correctness.

Values are restored in dependency order, and their parent-version records are updated. Two scopes can produce the same masked canvas but different cache contents and reuse permissions. Their future action values can therefore differ. Consistency with $\widehat{G}$ does not show that the graph captures every influence in the language model.

3.6 Online Inference

At each checkpoint, score the feasible candidates in $\mathcal{C}(s)$ and let $\tilde{a}$ maximize $\widehat{V}(s, a)$ over that pool. For a fixed margin $\delta \geq 0$ chosen on development data, execute

$$\hat{a} = \tilde{a} \text{ if } \widehat{V}(s, \tilde{a}) > \delta; \qquad \hat{a} = a_0 \text{ otherwise.} \tag{8}$$

Apply selective retention, advance U, and repeat without running alternative continuations. Fallback rules, repair caps, and a final checkpoint within the budget limit execution. When the budget runs out, return a parsed answer or record failure (Appendix C.6).

4 THEORETICAL ANALYSIS

We analyze scope advantage, learned selection, and candidate omission under a fixed continuation. Appendix A gives proofs. Appendices A.4 and A.6 cover cache isolation and constructed dependency examples.

4.1 ACTION VALUES AND REPAIR ORACLES

Fix the state s, continuation π, remaining budget b, executor, advancement rule, and positive cost scale $\kappa(s)$. Write $Q(s,a) = P(s,a) - \lambda C(s,a)$ as in Equation 3. Cost includes all charged computation after this decision. Earlier work is shared across actions. Over the feasible induced-action space and its nonempty candidate pool, define

$$a^*(s) \in \arg\max_{a \in \mathcal{A}(s)} Q(s,a), \qquad a^*_{\mathcal{C}}(s) \in \arg\max_{a \in \mathcal{C}(s)} Q(s,a). \tag{9}$$

These oracles maximize expected value under the fixed continuation. Relative values $V(s,a) = Q(s,a) - Q(s,a_0)$ preserve rankings and gaps (Appendix A.1). The full-closure action for target v is $a_{\mathrm{cl}} = a_{D_s(\{v\})}$; comparisons below assume it is feasible.

4.2 WHY FULL CLOSURE CAN BE SUBOPTIMAL

Graph paths show which units may affect others. Whether recomputing a descendant is useful depends on the current state. Selective repair can save work, but must include the cost of checking reuse and its effect on final reward.

Proposition 1 (Repair-scope advantage)**.** *Let a_S be a feasible action that retains one descendant for verification while a_{cl} regenerates it. Closure has expected regeneration cost g_C. Selective repair pays expected verification cost v and extra overhead h. If reuse is rejected, regeneration has expected cost g_F. Let p be the probability of accepted reuse and $\alpha \le p$ the probability that wrong content is accepted for reuse. Suppose all other expected costs agree. Assume the continuations can be coupled with final reward loss at most $L \in [0,1]$ on erroneous reuse and no loss otherwise. In common normalized cost units,*

$$Q(s,a_S) - Q(s,a_{\mathrm{cl}}) \ge \lambda\big[g_C - v - h - (1-p)g_F\big] - \alpha L. \tag{10}$$

If $g_C = g_F = g$, $\lambda(pg - v - h) > \alpha L$ guarantees a strict advantage.

Every check is charged. $1-p$ denotes rejection, not erroneous reuse. With equal expected reward and $g_C = g_F = g$, the difference is exactly $\lambda(pg - v - h)$ and can reverse sign across states sharing one graph. Appendix A.2 proves the bound, gives an example of this reversal, and includes all downstream costs.

4.3 LEARNED SELECTION AND ADVANTAGE OVER CLOSURE

Proposition 2 (Candidate selection guarantee)**.** *For a finite candidate pool containing a_0, suppose $|\widehat{V}(s,a) - V(s,a)| \le \epsilon$ for all $a \in \mathcal{C}(s)$. If $\hat{a} \in \mathcal{C}(s)$ satisfies $\widehat{V}(s,\hat{a}) \ge \max_{a \in \mathcal{C}(s)} \widehat{V}(s,a) - \delta$, where $\delta \ge 0$, then*

$$Q(s,a^*_{\mathcal{C}}) - Q(s,\hat{a}) \le 2\epsilon + \delta. \tag{11}$$

Equation 8 meets this condition: it selects the highest-scoring action, or the zero-score baseline when the maximum is at most δ. Uniform head errors ϵ_r, ϵ_c give $\epsilon = \epsilon_r + \lambda\epsilon_c$.

Corollary 1 (When learned repair outperforms closure)**.** *If $a_{\mathrm{cl}} \in \mathcal{C}(s)$, define the candidate opportunity $\Delta_{\mathrm{cl}}(s) = Q(s,a^*_{\mathcal{C}}) - Q(s,a_{\mathrm{cl}}) \ge 0$. Under Proposition 2,*

$$Q(s,\hat{a}) - Q(s,a_{\mathrm{cl}}) \ge \Delta_{\mathrm{cl}}(s) - 2\epsilon - \delta. \tag{12}$$

Thus $\Delta_{\mathrm{cl}}(s) > 2\epsilon + \delta$ ensures strict improvement.

Useful candidates need gains that exceed prediction error and the deployment margin. Appendix A.3 proves both results and analyzes margins, head errors, and gaps between action values. Training loss does not guarantee the assumed uniform value accuracy.

4.4 Candidate coverage and total decision regret

Proposition 3 (Coverage–selection decomposition)**.** *Define candidate omission by* $\eta_{\mathcal{C}}(s) = Q(s, a^*) - Q(s, a^*_{\mathcal{C}}) \geq 0$*. Under Proposition 2,*

$$Q(s, a^*) - Q(s, \hat{a}) = \eta_{\mathcal{C}}(s) + \left[Q(s, a^*_{\mathcal{C}}) - Q(s, \hat{a})\right] \leq \eta_{\mathcal{C}}(s) + 2\epsilon + \delta. \tag{13}$$

With fixed action values, larger nested pools cannot increase the loss from missing candidates. Improving the scorer targets selection error. Pool size alone does not bound omission. This result (Appendix A.3) motivates separate coverage and scorer ablations. Empirical reference gaps assess selection within a finite pool, not the best feasible value or uniform error. Policy changes require updated continuation values.

The interaction in Equation 4 also limits probability-additive reward approximation. Appendix A.5 gives this restricted lower bound. Nonlinear additive-logit predictors are a separate control in the experiments.

5 Experiments

Our research questions test how state affects repair value (RQ1) and how quality trades off against computation (RQ2). RQ3 tests the roles of candidate coverage, action scoring, and selective retention. Other controls test graph source and deployment-policy changes.

Experimental setup. We evaluate LLaDA-8B-Instruct, iLLaDA-8B-Instruct, and Dream-v0-Instruct-7B (Nie et al., 2025; 2026; Ye et al., 2025) on GSM8K, MATH, HumanEval, and MBPP (Cobbe et al., 2021; Hendrycks et al., 2021; Chen et al., 2021; Austin et al., 2021b). Their full test sets contain 1,319, 5,000, 164, and 500 problems. Mathematics uses final-answer accuracy and code uses pass@1. Runs use one NVIDIA GeForce RTX 5090 GPU with 32 GB memory, batch size one, and three seeds. Problems are split before state generation; test problems are excluded from training and hyperparameter selection. Each training state supplies at most six candidates and two stochastic continuations per state–action pair. Controller settings are fixed after selection on development data. Appendix C gives implementation and termination details.

Original uses native decoding. **Target-only-visible** reopens the target with descendants visible. **Full closure** regenerates the predicted affected region. **Confidence** uses confidence signals. **PRICE** learns complete-action value with selective retention. The valid target-only action instead quarantines $D_s(\{v\})$. Table 1 uses native termination. A shared 40-NFE protocol compares PURE-style, T2M-style, MDM-VGB-style, and UnMaskFork-style adaptations (Appendix B.2). Source-protocol scores appear in Appendix B.1.

NFE counts every dense backbone forward, including planning, denoising, regeneration, and verification (Shah et al., 2026). Latency also includes scoring, candidate construction, state/cache maintenance, and answer extraction. Main-table costs use four-task macro-averages: $\overline{\mathrm{NFE}} = \frac{1}{4}\sum_t \mathrm{NFE}_t$ and $\overline{L}_{\mathrm{rel}} = \frac{1}{4}\sum_t L_{\mathrm{method},t}/L_{\mathrm{Original},t}$, where t indexes the four tasks and $L_{\mathrm{method},t}$ is mean task latency, averaged over examples and seeds. State-level utility uses normalized remaining NFE and $\lambda = 0.20$ (Appendix B.4).

Main results and budget-matched inference. PRICE scores higher than full closure on all twelve backbone–task pairs (Table 1), with 12.0–12.3% fewer macro-averaged NFE. On LLaDA, GSM8K/MATH improve from $80.4\%/44.5\%$ to $82.6\%/46.8\%$; relative latency falls from 1.28 to 1.13. Across backbones, scores exceed Original at 1.12–1.13 times its latency. The separate 40-NFE reruns (Table 2) share a frozen backbone and count every call, including discarded branches. Controls labelled "style" are adapted reimplementations. PRICE exceeds T2M-style by 0.7–0.9 points and UnMaskFork-style by 0.2–0.5 points, using 36.4 versus 37.6 and 39.1 NFE.

State-conditioned repair opportunity. We compare closure, a *Best Fixed Scope* chosen on development data, a *Graph-only Selector* with structural features, and PRICE. The empirical reference $\tilde{a}_{\mathcal{C}}$ maximizes utility on the selection samples. Disjoint evaluation continuations estimate utility u_a for each evaluated action. At the same state, define

$$\widehat{\Delta}_{\mathrm{opp}} = u_{\tilde{a}_{\mathcal{C}}} - u_{a_{\mathrm{cl}}}, \qquad \widehat{\Delta}_{\mathrm{PRICE}} = u_{\hat{a}} - u_{a_{\mathrm{cl}}}, \qquad \widehat{G}_{\mathrm{ref}} = \widehat{\Delta}_{\mathrm{opp}} - \widehat{\Delta}_{\mathrm{PRICE}}. \tag{14}$$

Table 1: Native-termination results: three-seed means on instruction-tuned backbones. Scores are accuracy/pass@1 (%); HE denotes HumanEval. NFE and relative latency are four-task macro-averages; latency is normalized to Original on the same backbone and GPU.

Backbone	Method	GSM8K	MATH	HE	MBPP	NFE ↓	Rel. lat. ↓
LLaDA-8B	Original	77.5	42.2	49.4	41.0	32.0	1.00
	Target-only-visible	78.6	43.1	50.6	42.0	34.1	1.06
	Full closure	80.4	44.5	52.4	43.1	41.3	1.28
	Confidence	80.0	44.1	51.8	42.8	36.8	1.14
	PRICE	**82.6**	**46.8**	**54.9**	**45.3**	**36.2**	**1.13**
iLLaDA-8B	Original	89.0	56.7	65.9	58.0	32.0	1.00
	Target-only-visible	89.4	57.4	66.7	58.6	33.8	1.05
	Full closure	90.2	58.5	68.1	59.6	40.7	1.27
	Confidence	90.0	58.1	67.7	59.2	36.1	1.13
	PRICE	**91.1**	**60.3**	**69.9**	**61.2**	**35.7**	**1.12**
Dream-7B	Original	81.0	39.2	55.5	58.8	32.0	1.00
	Target-only-visible	81.8	40.1	56.7	59.5	34.0	1.06
	Full closure	83.2	41.5	58.1	60.8	40.9	1.27
	Confidence	82.8	41.1	57.7	60.4	36.5	1.14
	PRICE	**84.7**	**43.4**	**60.4**	**62.5**	**36.0**	**1.13**

Table 2: Matched 40-NFE LLaDA reruns. Scores (%); NFE averages all four tasks.

Method	GSM8K	MATH	HE	MBPP	NFE ↓
Original	78.1	42.8	50.2	41.8	32.0
Confidence	80.0	44.1	51.8	42.8	36.8
PURE-style	80.5	44.6	52.2	43.3	37.2
T2M-style	80.8	44.9	52.6	43.5	37.6
MDM-VGB-style	81.0	45.2	52.8	43.8	38.0
UnMaskFork-style	81.2	45.4	53.0	44.0	39.1
Full closure	79.9	44.0	51.8	42.7	40.0
PRICE	**81.5**	**45.7**	**53.5**	**44.2**	**36.4**

Table 3: LLaDA scorer ablation. NFE is the GSM8K/MATH mean.

Scorer	GSM8K	MATH	NFE ↓
Fixed Closure	80.4	44.5	41.5
Random	79.5	43.8	38.9
Confidence	80.7	44.6	37.2
Additive Probability	81.1	45.1	37.0
Additive Logit	81.7	45.8	36.9
Joint Scorer	82.2	46.4	36.8
Joint + Paired (PRICE)	82.6	46.8	36.7
Candidate Oracle	84.1	48.7	–

Table 4: State-level selection on LLaDA.

Policy	Utility	Gain	Gap	Agree.
Full Closure	0.711	0.000	0.071	21.0%
Best Fixed	0.736	0.025	0.046	35.8%
Graph-only	0.750	0.039	0.032	49.9%
PRICE	0.765	0.054	0.017	68.5%
Best-in-pool	0.782	0.071	0.000	100.0%

Table 5: Reuse and graph diagnostics (%).

Task	Accept	Harm	Recall
GSM8K	63.8	1.32	84.5
MATH	59.4	2.94	82.9
HE	66.2	1.98	86.7
MBPP	64.7	2.18	85.8

Table 6: Matched execution on LLaDA/GSM8K (accuracy, %).

Execution	Acc.	NFE ↓
Visible stale	79.8	34.6
Full regen.	80.4	40.9
Selective reuse	82.6	36.0

All three tables use LLaDA. Selection averages four tasks and uses normalized remaining NFE at $\lambda = 0.20$; reference selection and evaluation use disjoint samples. Harm counts reuse failure with successful paired regeneration among accepted checks; recall measures coverage of intervention-derived invalidation. Execution uses matched decision states and controls; selective reuse implements selective retention.

Utility gaps share the theoretical Q scale. The reference gains 0.071 over closure; fixed scopes, structural selection, and PRICE capture 0.025, 0.039, and 0.054, respectively (Table 4). The 0.017 gap measures empirical within-pool selection loss. Per-task results and scope frequencies appear in Appendix B.5.

Candidate coverage. In the enumeration analysis (Table 13; Appendix B.7) on 1,000 shared states, the current pool captures 70.5% of enumeration opportunity (0.778 vs. 0.806 utility; baseline 0.711), with omission/selection gaps $0.028/0.013$ (Table 14). Raising the cap from 6 to 24 increases

selected utility from 0.765 to 0.784 and scoring cost from 0.49 to 1.47 ms/state (Appendix B.7). Its offline reference utility rises to 0.803, close to enumeration's 0.806, indicating broader candidate coverage. Enumeration is offline; deployment defaults to at most six actions. The controller captures 56.8% of enumeration opportunity and 80.6% within its pool, separating construction and selection gaps.

Action-value estimation. Under shared candidates (Table 3), additive-logit scoring reaches 81.7%/45.8% on GSM8K/MATH, joint scoring 82.2%/46.4%, and joint paired training 82.6%/46.8%. The paired comparison matches encoder and continuation-call count. Its outcome-informed reference scores 84.1%/48.7%. NFE averages GSM8K/MATH. Additive-logit scoring allows nonlinear reward interactions. Appendix B.8 separates controls for paired collection and regression on relative targets.

Verification-gated reuse. With states, selected scope, continuation, budgets, and seeds matched, selective retention reaches 82.6% on GSM8K at 36.0 NFE, versus 80.4%/40.9 for full regeneration and 79.8%/34.6 for visible-stale execution (Table 6). Checks accept 59.4%–66.2% of attempts. Of accepted cases, 1.32%–2.94% fail while paired regeneration succeeds (Table 5). Cases where both fail are reported separately. A task-aware checker gains 0.2 accuracy points for 1.2 more NFE (Appendix B.15), using symbolic/numerical mathematics checks and public code tests without benchmark answers or hidden tests. Appendix B.15 gives counts and local costs. Checks cost one NFE each; savings including rejection continuation are 0.88–1.13 NFE/attempt (Table 26).

Graph contribution and policy shift. Graph-source controls (Appendix B.10) fix the scorer and executor. Replacing a random graph with the predicted graph raises GSM8K accuracy from 80.4% to 82.6%; the trace-derived approximation adds 0.4 points. At 20% edge removal/addition, accuracy is 81.4%/81.7% and NFE is 35.5/37.6, respectively (Appendix B.9). Intervention-derived invalidation recall is 82.9%–86.7% (Table 5); precision is 86.1%–89.2% (Appendix B.11, with accuracy gaps within recall groups and quarantine costs). Refresh retrains the scorer on training states from the deployment policy; two rounds raise agreement from 65.8% to 70.6% and utility from 0.741 to 0.758. The rounds add 54.9 and 56.9 offline GPU-hours, respectively (Appendices B.12–B.13).

Offline cost and variability. Initial collection uses 120,000 states and 1.35 million continuations, requiring 224.6 GPU-hours; scorer training adds 6.4 GPU-hours. Each refresh adds 30,000 states. Collection and training require $53.1 + 1.8$ and $55.0 + 1.9$ GPU-hours for the two rounds on the same GPU (Appendix B.13). These offline costs are shared by requests using one controller and are separate from online inference. Under native termination, three-seed accuracy (mean $\pm$ standard deviation) is $82.6 \pm 0.28\%$ for PRICE versus $80.4 \pm 0.31\%$ for closure on GSM8K, and $46.8 \pm 0.36\%$ versus $44.5 \pm 0.42\%$ on MATH (Appendix B.3).

Quality–cost operating points. Increasing λ from 0 to 0.80 reduces macro NFE from 39.4 to 33.5 and relative latency from 1.21 to 1.05, while GSM8K/MATH change from 83.0%/47.0% to 80.9%/45.0% (Appendix B.14). Task-specific accuracy–latency points and costs appear in Figure 2 and Appendix B.16.

6 Limitations

Gains depend on candidate value under the current continuation. Smaller scopes may not save dense forwards. Checks, planning, and state costs can cancel reuse savings. Graph errors affect candidates and execution. Reward optimization may cause premature stopping or repeated revision. Bounds assume fixed continuation values and uniform prediction accuracy, which training loss does not guarantee. Past effects, false acceptance, and distribution shift go beyond structural consistency.

7 Conclusion

PRICE learns repair-scope reward–cost values from matched offline interventions. It runs one action with quarantine and checked reuse. Theory bounds the effects of candidate coverage and value error. Experiments improve scores over full closure with fewer model evaluations.

REPRODUCIBILITY

Section 5 specifies hardware, evaluation sizes, seed aggregation, candidate caps, and continuation counts. Appendix B reports the adapted baseline protocols, native LLaDA seed variability, state-level comparisons, paired reuse outcomes, graph diagnostics, and online and offline costs. Appendix C details execution, verification, and termination. Code is provided in the supplementary materials.

AI USE STATEMENT

During the preparation of this manuscript, the authors used large language models (LLMs) to assist with research ideation, literature exploration, mathematical proof checking, and manuscript refinement. The tools were used to support research discussions, improve the clarity of technical descriptions, assist in organizing and refining manuscript content, and identify potential issues in mathematical derivations and formulations.

REFERENCES

Marianne Arriola and Volodymyr Kuleshov. Set diffusion: Interpolating token orderings between autoregression and diffusion for fast and flexible decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2607.01775.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, 2021a. URL https://proceedings.neurips.cc/paper/2021/hash/958c530554f78bcd8e97125b70e6973d-Abstract.html.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b. URL https://arxiv.org/abs/2108.07732.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas S. Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/510e22f4a2da5212a64f1591736e2eaf-Abstract-Conference.html.

Akash Bonagiri, Devang Borkar, Gerard Janno Anderias, Setareh Rafatirad, and Houman Homayoun. CausalFlow: Causal attribution and counterfactual repair for LLM agent failures. *arXiv preprint arXiv:2605.25338*, 2026. URL https://arxiv.org/abs/2605.25338.

Shidong Cao, Hongzhan Lin, Yuxuan Gu, Ziyang Luo, and Jing Ma. DiffCoT: Diffusion-styled chain-of-thought reasoning in LLMs. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 38942–38957. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1939. URL https://aclanthology.org/2026.findings-acl.1939/.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Tianlang Chen, Minkai Xu, Jure Leskovec, and Stefano Ermon. RFG: Test-time scaling for diffusion large language model reasoning with reward-free guidance. *arXiv preprint arXiv:2509.25604*, 2025. URL https://arxiv.org/abs/2509.25604.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

C. Nicolò De Sabbata, Theodore R. Sumers, Badr AlKhamissi, Antoine Bosselut, and Thomas L. Griffiths. Rational metareasoning for large language models. *arXiv preprint arXiv:2410.05563*, 2024. URL https://arxiv.org/abs/2410.05563.

Andrey Fomenko, Maksim Kryzhanovskiy, Svetlana Glazyrina, and Roman Ischenko. NAVIRA: Decoupled stochastic remasking for masked diffusion language models. *arXiv preprint arXiv:2606.06031*, 2026. URL https://arxiv.org/abs/2606.06031.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. DiffuCoder: Understanding and improving masked diffusion models for code generation. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/1a5b00e59818e4be1c89855a8b921903-Abstract-Conference.html.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't settle too early: Self-reflective remasking for diffusion language models. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5d8bc24ef15b7a808b6dc403c01583e2-Abstract-Conference.html.

Daniel Israel, Tian Jin, Ellie Cheng, Guy Van den Broeck, Aditya Grover, Suvinay Subramanian, and Michael Carbin. Planned diffusion. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=wZN8debH4W.

Metod Jazbec, Theo X. Olausson, Louis Béthune, Pierre Ablin, Michael Kirchhof, João Monteiro, Victor Turrisi, Jason Ramapuram, and Marco Cuturi. Learning unmasking policies for diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2512.09106.

Kijung Jeon, Thuy-Duong Vuong, and Molei Tao. VGB for masked diffusion model: Efficient test-time scaling for reward satisfaction and sample editing. *arXiv preprint arXiv:2606.28301*, 2026. URL https://arxiv.org/abs/2606.28301.

Bumjun Kim, Dongjae Jeon, Moongyu Jeon, and Albert No. DAPD: Dependency-aware parallel decoding via attention for diffusion LLMs. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.12996.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32819–32848. PMLR, 2024. URL https://proceedings.mlr.press/v235/lou24a.html.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Roy Miles, Aysim Toker, Andreea-Maria Oncescu, Songcen Xu, Jiankang Deng, and Ismail Elezi. Test-time scaling with diffusion language models via reward-guided stitching. *arXiv preprint arXiv:2602.22871*, 2026. URL https://arxiv.org/abs/2602.22871.

Kou Misaki and Takuya Akiba. UnMaskFork: Test-time scaling for masked diffusion via deterministic action branching. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.04344.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/48b383b24230e0e6e649d9c98dae4d8c-Abstract-Conference.html.

Shen Nie, Qiyang Min, Shaoxuan Xu, Zihao Huang, Yuxuan Song, Yong Shan, Yankai Lin, Wayne Xin Zhao, Chongxuan Li, and Ji-Rong Wen. Improved large language diffusion models. *arXiv preprint arXiv:2606.25331*, 2026. URL https://arxiv.org/abs/2606.25331.

Yangryeol Park, Kunhui Lee, Hanback Choi, Cheoneum Park, Donghyeon Jeon, Inho Kang, and Seung-Hoon Na. PURE: Post-hoc unlocking and REfinement for discrete diffusion decoding. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 24653–24667. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1234. URL https://aclanthology.org/2026.findings-acl.1234/.

Houxing Ren, Mingjie Zhan, Zimu Lu, Ke Wang, Yunqiao Yang, Haotian Hou, Junting Pan, and Hongsheng Li. Edit-based refinement for parallel masked diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.09603.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb0b13cc515724ab8015bc978fdde0ad-Abstract-Conference.html.

Yash Shah, Abhijit Chakraborty, and Vivek Gupta. CaRE compute-aware remasking evaluation protocol for masked diffusion language models. *arXiv preprint arXiv:2607.24763*, 2026. URL https://arxiv.org/abs/2607.24763.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2503.00307.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *International Conference on Learning Representations*, 2026a. URL https://arxiv.org/abs/2505.22618.

Yifan Wu, Yufeng Zhang, and Kenli Li. CORA-Diff: Confidence-oriented residual acceptance for efficient diffusion language model inference. *arXiv preprint arXiv:2608.11235*, 2026b. URL https://arxiv.org/abs/2608.11235.

Lin Yao. Remask, don't replace: Token-to-mask refinement in diffusion language models. *arXiv preprint arXiv:2604.18738*, 2026. URL https://arxiv.org/abs/2604.18738.

Jiacheng Ye, Shansan Gong, Liheng Chen, Lin Zheng, Jiahui Gao, Han Shi, Chuan Wu, Xin Jiang, Zhenguo Li, Wei Bi, and Lingpeng Kong. Diffusion of thought: Chain-of-thought reasoning in diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/be30024e7fa2c29cac7a6dafcbb8571f-Abstract-Conference.html.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025. URL https://arxiv.org/abs/2508.15487.

Longxuan Yu, Shaorong Zhang, Yu Fu, Hui Liu, Yue Dong, and Greg Ver Steeg. Revise, don't freeze: Sampler-matched training for self-correcting masked diffusion language models. *arXiv preprint arXiv:2606.01026*, 2026. URL https://arxiv.org/abs/2606.01026.

Shuibai Zhang, Fred Zhangzhi Peng, Yiheng Zhang, Jin Pan, and Grigorios G. Chrysos. Corrective diffusion language models. *arXiv preprint arXiv:2512.15596*, 2025. URL https://arxiv.org/abs/2512.15596.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1898. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/52190a0362148d179f1cbd9080956872-Abstract-Conference.html.

Hao Zou, Zachary Horvitz, Chandhru Karthick, Zhou Yu, and Kathleen McKeown. Detect, remask, repair: Diffusion editing for faithful summarization of evolving contexts. *arXiv preprint arXiv:2606.12807*, 2026. URL https://arxiv.org/abs/2606.12807.

A Proofs and Counterexamples

A.1 State normalization and relative repair utility

Fix a state and a normalizer $\kappa(s) > 0$ shared by all actions. The normalized expected cost, relative repair cost, and relative utility satisfy

$$C(s,a) = \mu_C(s,a)/\kappa(s), \qquad \Delta c_a = [\mu_C(s,a) - \mu_C(s,a_0)]/\kappa(s),$$
$$U_\lambda(s,a) = P(s,a) - \lambda \Delta c_a = Q(s,a) + \lambda C(s,a_0).$$

The last term is constant across candidates. Thus $U_\lambda(s,a) - U_\lambda(s,a') = Q(s,a) - Q(s,a')$, so the scope-benefit, coverage, and selection results also hold for relative utility. With $\kappa(s) = \mu_C(s,a_0) > 0$, the additive constant is λ. All candidates share this denominator; none uses its own continuation cost. If $\mu_C(s,a_0) = 0$, fix a positive $\kappa(s)$; then $U_\lambda(s,a) = Q(s,a)$.

A.2 Repair-scope advantage and complete remaining cost

Proof of Proposition 1. Write the common normalized expected cost of all other work as C_{base}. Taking the conditional expectation over acceptance and rejection gives

$$C(s,a_{\text{cl}}) = C_{\text{base}} + g_C, \qquad C(s,a_S) = C_{\text{base}} + v + h + (1-p)g_F.$$

For $p = 1$, rejection contributes zero and g_F need not be specified. Let E denote acceptance of incorrect reuse in the coupled continuations, with $\Pr(E \mid s) = \alpha$. Write r_{cl}, r_S for their terminal rewards. The assumption $r_{\text{cl}} - r_S \le L\mathbf{1}_E$ gives $P(s,a_{\text{cl}}) - P(s,a_S) \le \alpha L$. Subtracting action values gives

$$\begin{aligned} Q(s,a_S) - Q(s,a_{\text{cl}}) &= P(s,a_S) - P(s,a_{\text{cl}}) + \lambda\big[C(s,a_{\text{cl}}) - C(s,a_S)\big] \\ &\ge \lambda\big[g_C - v - h - (1-p)g_F\big] - \alpha L. \end{aligned}$$

For $g_C = g_F = g$, the saving is $pg - v - h$. This gives the condition for strict advantage in Proposition 1. If expected terminal rewards agree, their difference is zero and the utility difference is exactly $\lambda(pg - v - h)$. □

The coupling is a sufficient reward-preservation assumption. The consistency probe does not guarantee it. This assumption can be weakened to the corresponding conditional expected-loss bounds on E and its complement. Acceptance probability p, false-acceptance probability α, and probability that cached content remains valid are different. A rejected check triggers regeneration and is not false acceptance. Accepted text can agree with a model prediction and still be wrong.

All costs use one positive denominator $\kappa(s)$. For NFE, a one-pass check has $v = 1/\kappa(s)$. For g_C and g_F, divide the expected remaining regeneration NFE by the same scale. Extra backbone calls enter h after normalization. Controller and cache overhead that requires no forward calls enters only the time metric. Under time accounting, all four terms use normalized time. If a rejected probe initializes later denoising, g_F counts only the additional work, so the probe is not charged twice. Include all downstream costs that depend on the action; a common C_{base} requires equal expected costs for that work.

State-dependent preference reversal. Consider two states with the same dependency graph and equal expected reward across the two actions. Let both states have the same normalized $g > 0, v, h$, with $g_C = g_F = g$. If their acceptance probabilities satisfy $p_+ > (v+h)/g > p_-$, selective repair is strictly better at the first state and full closure is strictly better at the second, for $\lambda > 0$. Thus the graph alone may not determine the preferred scope. This supports choosing the scope based on state without specifying how to estimate its value.

More generally, let $\Delta P = P(s,a_S) - P(s,a_{\text{cl}})$ and $\Delta C = C(s,a_{\text{cl}}) - C(s,a_S)$. The identity $Q(s,a_S) - Q(s,a_{\text{cl}}) = \Delta P + \lambda \Delta C$ holds even when the matched-cost or reward-preservation assumptions fail. Let G be the random avoided cost, J the reuse indicator, and T, H the verification and overhead costs, where G, T, H use raw cost units. A matched-cost coupling gives $\Delta C = \mathbb{E}[JG - T - H]/\kappa(s)$. In general $\mathbb{E}[JG] \ne \Pr(J=1)\mathbb{E}[G]$. The full action values include budget exhaustion and changes in downstream work.

A.3 Relative values, candidate omission, and deployment margin

Proof of Proposition 2. Choose $a_{\mathcal{C}} \in \arg\max_{a \in \mathcal{C}(s)} Q(s,a)$. It also maximizes $V(s,a)$ because all actions subtract the same baseline. Write $V_a = V(s,a)$ and $\widehat{V}_a = \widehat{V}(s,a)$. Then

$$\begin{aligned} Q(s,a_{\mathcal{C}}) - Q(s,\hat{a}) &= V_{a_{\mathcal{C}}} - V_{\hat{a}} \\ &= (V_{a_{\mathcal{C}}} - \widehat{V}_{a_{\mathcal{C}}}) + (\widehat{V}_{a_{\mathcal{C}}} - \widehat{V}_{\hat{a}}) + (\widehat{V}_{\hat{a}} - V_{\hat{a}}) \le 2\epsilon + \delta. \end{aligned}$$

This proves Equation 11. □

To check the margin rule, let a_g maximize $\widehat{V}$ over the same feasible candidate pool. If $\widehat{V}(s,a_g) > \delta$, executing a_g has no predicted selection gap. Otherwise, executing a_0 has gap $\widehat{V}(s,a_g) - \widehat{V}(s,a_0) \le \delta$, since the baseline score is exactly zero. Both satisfy the proposition. With $\delta = 0$, the rule is greedy, with ties resolved as specified. Uniform error bounds for both relative-value heads give

$$|\widehat{V} - V| \le |\widehat{\Delta r} - \Delta r| + \lambda |\widehat{\Delta c} - \Delta c| \le \epsilon_r + \lambda \epsilon_c,$$

for $\lambda \ge 0$. The cost-head error ϵ_c uses normalized units $C_{\mathrm{raw}}/\kappa(s)$. Any output scaling used only in training is undone before scoring; the state normalization $\kappa(s)$ is kept. If the head instead predicts a raw-cost difference with error at most ϵ_{raw}, its contribution to the value-error bound is $\lambda \epsilon_{\mathrm{raw}}/\kappa(s)$. For a shared estimated normalizer, the same algebra holds conditional on that estimate. The bound does not control normalizer estimation error.

If the best candidate is unique and its value exceeds every other candidate by a gap $\gamma > 2\epsilon + \delta$, choosing another action would contradict the candidate-pool regret bound. Without this gap, value error tending to zero need not lead to one fixed maximizer: equally good actions may alternate. Even perfect ranking cannot recover omitted candidates. The bound does not control distribution shift when the controller changes the visited states or continuation policy. All expectations use the same executor, verifier, cost definition, and remaining budget.

Proof of Corollary 1. Since closure is in the pool, $\Delta_{\mathrm{cl}}(s) \ge 0$. Also,

$$Q(s,\hat{a}) - Q(s,a_{\mathrm{cl}}) = \Delta_{\mathrm{cl}}(s) - \big[Q(s,a^*_{\mathcal{C}}) - Q(s,\hat{a})\big] \ge \Delta_{\mathrm{cl}}(s) - 2\epsilon - \delta,$$

by Proposition 2. A positive lower bound gives strict improvement over closure. □

Proof of Proposition 3. Adding and subtracting $Q(s,a^*_{\mathcal{C}})$ gives

$$Q(s,a^*) - Q(s,\hat{a}) = \big[Q(s,a^*) - Q(s,a^*_{\mathcal{C}})\big] + \big[Q(s,a^*_{\mathcal{C}}) - Q(s,\hat{a})\big].$$

The first term is $\eta_{\mathcal{C}}(s)$; Proposition 2 bounds the second. For nested pools with fixed action values, the larger pool's maximum cannot decrease, so omission cannot increase. □

A.4 Isolation at a replacement call

Proposition 4 (Invariance to quarantined cache payloads)**.** *Fix an action and a replacement call before cached text is restored. Let ξ contain the current visible input and control metadata for that call. Assume clearing removes the quarantined text from all generator inputs and invalidates any KV or hidden-state cache containing that text. Assume the generator cannot read the external cache H. For caches H, H' that differ only in quarantined text, the replacement kernel satisfies*

$$K_\theta(\cdot \mid \xi, a, H) = K_\theta(\cdot \mid \xi, a, H'). \tag{15}$$

Proof of Proposition 4. Under the clearing and cache-invalidation assumptions, all generator inputs at this call depend only on ξ and the fixed action a. H is not an input. Thus the replacement kernel can be written as $K_\theta(\cdot \mid \xi, a, H) = \widetilde{K}_\theta(\cdot \mid \xi, a)$. Replacing H by H' leaves the right-hand side unchanged, proving the claim. Equivalently, coupling the two calls with the same random draws gives identical replacement samples. □

Clearing must remove the text from every generator input path, including cached keys and values, hidden representations, and any copy sent back to the generator. Changing a stale flag while keeping token identities does not meet this assumption. The result concerns invariance to the cached text; it does not require zero attention to positions. A masked slot still has a position and a mask embedding. The result also conditions on the selected action and current generator inputs. It does not compare full trajectories where changing cached text changes verification decisions, reuse timing, control decisions, or remaining budget. Restoring accepted cached text makes it an input again, so the equality need not hold. An unsound verifier or an incomplete predicted graph can still cause incorrect generation.

A.5 Restricted additive lower bound

For executable scopes with a fixed advancement rule, executor, reuse policy, continuation, and budget, write $F_s(R) = P_b^\pi(s, a_R)$ and define

$$\mathcal{I}_s^P(A, B) = F_s(A \cup B) - F_s(A) - F_s(B) + F_s(\varnothing). \tag{16}$$

Positive interaction means that the scopes are complementary in measured reward; it need not indicate a semantic dependency. Independent errors under an all-or-nothing reward can produce the same pattern. Cost has a similar interaction, and net-value interaction equals $\mathcal{I}_s^P - \lambda \mathcal{I}_s^C$.

Proposition 5 (Restricted additive approximation)**.** *For disjoint A, B, consider a predictor strictly additive in probability space, $\widehat{F}_s(R) = c_s + \sum_{i \in R} v_i(s)$, with state intercept c_s and unit contributions $v_i(s)$ fixed across scopes. Its maximum absolute error is at least $|\mathcal{I}_s^P(A, B)|/4$ on $\varnothing, A, B, A \cup B$.*

Proof. Let $e_R = F_s(R) - \widehat{F}_s(R)$, with $v_i(s)$ fixed across the four scopes. Since the scopes are disjoint,

$$\widehat{F}_s(A \cup B) - \widehat{F}_s(A) - \widehat{F}_s(B) + \widehat{F}_s(\varnothing) = 0.$$

Thus $\mathcal{I}_s^P = e_{A \cup B} - e_A - e_B + e_\varnothing$, and the triangle inequality gives $|\mathcal{I}_s^P| \le 4 \max_R |e_R|$. □

This result tests a restricted representation. It does not show that a particular action-encoder architecture is needed. The restriction applies to the output probability, not the logit. Features that depend on the set or nonlinear output maps can break the assumption. Let $z_A, z_B \in \{0, 1\}$ denote membership and let $g_k = \sigma(k(2z_A + 2z_B - 3))$, where $\sigma(z) = (1 + e^{-z})^{-1}$ is the sigmoid and $k > 0$ controls its slope. Its logits on $(0, 0), (1, 0), (0, 1), (1, 1)$ are $-3k, -k, -k, k$. Each error relative to AND is at most e^{-k}. Thus a model additive before the sigmoid can approximate this reward interaction arbitrarily well. Approximation error and decision regret also differ: even a misspecified predictor can rank the best action correctly.

A.6 Constructed dependency examples

Consider $x = 2 + 3$, $y = 2x$, and a terminal check requiring both stored results to be correct, starting from $x = 4, y = 8$. Consider an executor that holds unselected values fixed. The rewards for no repair, x only, y only using the current x, and jointly recomputing x, y are $0, 0, 0, 1$. This example illustrates reward interaction; it is not a trace of the quarantine-and-verification procedure. Full descendant repair already handles it by regenerating y after x. Scope selection can instead reuse a descendant such as $z = \mathbf{1}[x > 0]$, which remains 1 when x changes from 4 to 5. This requires the paid reuse procedure to satisfy the reward and cost conditions of Proposition 1. Recognizing continued validity at no cost would require an oracle, not an executable reuse policy.

B Additional Experimental Analyses

B.1 Sources and Protocols for Recent Baselines

RemeDi. We use the SFT+RL row of Table 1 in arXiv:2509.23653v1 (Huang et al., 2026). RemeDi adds training and a confidence predictor to LLaDA-8B-Instruct. This differs from the unchanged checkpoint in our fixed-backbone study. Its evaluation uses blocks of 32, greedy sampling,

Table 7: External revision baselines: source-reported scores (%). For PURE, k is the number of tokens unmasked per draft step and B is the refinement-step budget. Dashes mark unavailable results.

Method / configuration	GSM8K	MATH	Human Eval	MBPP
RemeDi, SFT+RL	89.1	52.9	73.2	59.4
PURE, $k = 1, B = 10$	71.70	28.84	–	–
PURE, $k = 1, B = 40$	74.27	31.33	–	–
T2M, Dream-7B	79.23	–	–	–
T2M, LLaDA-8B-Base	73.39	–	–	–

These source-reported results use their original configurations, not our protocol. Different checkpoints and budgets prevent a matched-compute ranking against Table 1. PURE appeared in Findings of ACL 2026; RemeDi and T2M scores use the cited arXiv versions. Source protocols follow below.

and one token unmasked per step. Maximum generation lengths are 2,048 on MATH and 1,024 on the other tasks. The paper evaluates the full 5,000-example MATH test set.

PURE. The two rows come from Table 1 of Park et al. (2026), using LLaDA-8B-Instruct with 256-token drafts. At $k = 1$, draft generation uses 256 forwards; adding $B = 10$ or $B = 40$ refinement steps gives 266 or 296 forwards. The matched draft scores in that paper are 61.41 on GSM8K and 25.68 on MATH. Its main table reports single-run results unless stated otherwise. These budgets and draft scores differ from our Original row. Gains must therefore be measured against PURE's draft baseline.

Token-to-Mask. We use the Dream-7B and LLaDA-8B-Base results in Table 2 of Yao (2026) (arXiv:2604.18738v3). Both use 256 generation tokens and 256 scheduled decoding steps; remasking can require additional fill steps. Dream uses temperature 0.1 and top-$p = 0.9$. LLaDA uses a single 256-token block and a base checkpoint. Their native GSM8K scores are 79.53 and 73.31, respectively. We omit the paper's HumanEval+ and MBPP+ results on other backbones because they differ from our code benchmarks.

Relation to the controlled reruns. Appendix B.2 evaluates adapted PURE/T2M policies under a shared checkpoint and NFE cap. The source-protocol results above remain separate. RemeDi also changes the trained model, making it a different system. DAPD (Kim et al., 2026) provides a dependency-aware decoding comparison, but its MATH500 results cannot replace full MATH results. We do not convert published step counts or NFE/token into our macro costs.

B.2 Refinement and Inference Controls under a Shared NFE Cap

Table 2 compares all methods in a separate rerun with the same frozen LLaDA-8B-Instruct checkpoint and a per-example cap of 40 NFE, counting every generation, repair, and verification forward. For methods that invoke planning, its forwards also count toward this cap. Actual use may be below the cap. PURE-style follows span-level reopening and refinement (Park et al., 2026); T2M-style follows token-to-mask reopening (Yao, 2026). Both use our evaluation wrapper and shared decoding and termination rules, with no additional backbone training. The style labels mark these adaptations. PURE already uses LLaDA-8B-Instruct, but its original draft budget and refinement schedule differ from this setting; T2M likewise uses its source decoding protocol. Appendix B.1 records the original configurations. MDM-VGB-style ranks candidate denoising states with an auxiliary reward/value estimator and selects the next transitions (Jeon et al., 2026). UnMaskFork-style branches from a shared partial state and scores alternative continuations (Misaki & Akiba, 2026). Both adapt the source mechanisms to the same frozen LLaDA-8B-Instruct checkpoint and 40-NFE protocol. NFE counts every dense backbone forward used for branch generation, auxiliary scoring, and verification, including branches later discarded. Latency also includes scorer computation and state management. These adapted controls compare state/trajectory selection with PRICE's complete repair actions and restoration only after successful verification.

Under this cap, PRICE improves over T2M-style by 0.7–0.9 points across tasks while using 36.4 rather than 37.6 mean NFE. It exceeds capped closure by 1.5–1.7 points at 9.0% fewer NFE. Relative

Table 8: Three-seed LLaDA accuracy (%) under native termination. Entries are means ± reported standard deviations.

Task	Full closure	PRICE
GSM8K	80.4 ± 0.31	82.6 ± 0.28
MATH	44.5 ± 0.42	46.8 ± 0.36

Table 9: Best-in-pool scope frequencies on LLaDA repair states (%). Each row sums to 100.

Task	None	Target	Partial	Full	Global
GSM8K	8.3	24.8	43.6	19.7	3.6
MATH	6.7	20.6	47.9	21.5	3.3
HumanEval	7.9	22.1	44.5	21.8	3.7
MBPP	8.6	23.4	42.7	21.1	4.2

Partial scopes win in 42.7%–47.9% of states; full closure wins in 19.7%–21.8%. Target denotes the valid quarantining action, distinct from Target-only-visible. The reference uses estimated continuation value. Table 4 complements these frequencies with value gains over fixed and structural controls.

to MDM-VGB-style and UnMaskFork-style, score gains are 0.4–0.7 and 0.2–0.5 points, with 4.2% and 6.9% fewer NFE. Table 1 uses native termination without the shared 40-NFE truncation; its 41.3 closure NFE and 36.2 PRICE NFE are separate operating points. No numerical row transfers between settings.

B.3 Seed Variability

Table 8 adds variation across three seeds to the native-termination means. The standard deviations describe variation between runs of each method; they are not standard errors of the paired method differences. HumanEval in both native and 40-NFE runs uses 164 completions per seed, for 492 in total. Percentages are rounded after pooling the passed and total counts across seeds.

B.4 State-Level Repair Utility

For a positive no-repair continuation cost $\mu_C(s, a_0)$, define

$$\bar{C}(s,a) = \frac{\mu_C(s,a) - \mu_C(s,a_0)}{\mu_C(s,a_0)}, \qquad U_\lambda(s,a) = P(s,a) - \lambda \bar{C}(s,a). \tag{17}$$

Here $\mu_C(s,a)$ is expected remaining NFE; all state-level utility tables use $\lambda = 0.20$. At the population level, this is a ratio of expectations. We first average each action's continuation costs, then take the ratio. All candidates from a state share one baseline estimate. We do not average ratios with a different denominator for each rollout. This repair-state metric differs from benchmark accuracy and macro-averaged latency. Subtracting reported relative latency from a benchmark score does not recover it. With $\kappa(s) = \mu_C(s,a_0) > 0$, it matches the relative-cost objective analyzed in Appendix A. This normalization gives a state-dependent raw-cost coefficient $\lambda/\kappa(s)$, rather than a fixed coefficient on raw cost.

B.5 State-Level Selection and Scope Preferences

State-level diagnostics use repair states from held-out problems in the evaluation distribution. Selection and evaluation continuations do not overlap. Using the normalized-NFE utility from Appendix B.4, the reference is $\tilde{a}_C(s) \in \arg\max_{a \in \mathcal{C}(s)} \overline{U}_{\mathrm{sel}}(s,a)$, where $\overline{U}_{\mathrm{sel}}$ estimates utility from the selection continuations. Separate evaluation continuations estimate the utilities of this action, closure, and each deployable policy. The Graph-only Selector removes hidden representations, confidence, entropy, and token-content features. It keeps structural scope and graph features. Best Fixed Scope uses a depth template chosen on development data: depth 1 on GSM8K, HumanEval, and MBPP, and depth 2 on MATH.

Table 10 gives the task values used in Table 4, which averages tasks equally. Displayed values are rounded separately. PRICE captures approximately 76% of the reference advantage. This is a ratio

Table 10: Repair utility by task on held-out LLaDA states. Gain is relative to closure; gap and agreement use the empirical best-in-pool reference. Independent continuations evaluate selected actions.

Policy	Utility	Gain	Gap	Agree.
GSM8K				
Full Closure	0.746	0.000	0.071	19.7%
Best Fixed	0.772	0.026	0.045	36.4%
Graph-only	0.786	0.040	0.031	51.2%
PRICE	0.800	0.054	0.017	70.4%
Best-in-pool	0.817	0.071	0.000	100%
MATH				
Full Closure	0.681	0.000	0.075	21.5%
Best Fixed	0.707	0.026	0.049	34.9%
Graph-only	0.721	0.040	0.035	47.8%
PRICE	0.738	0.057	0.018	66.7%
Best-in-pool	0.756	0.075	0.000	100%

Policy	Utility	Gain	Gap	Agree.
HumanEval				
Full Closure	0.704	0.000	0.070	21.8%
Best Fixed	0.728	0.024	0.046	35.6%
Graph-only	0.742	0.038	0.032	50.1%
PRICE	0.758	0.054	0.016	68.2%
Best-in-pool	0.774	0.070	0.000	100%
MBPP				
Full Closure	0.713	0.000	0.068	21.1%
Best Fixed	0.737	0.024	0.044	36.2%
Graph-only	0.749	0.036	0.032	50.4%
PRICE	0.765	0.052	0.016	68.7%
Best-in-pool	0.781	0.068	0.000	100%

of aggregate gains, not an average of ratios over states. Agreement measures how often the policy selects the same action as the empirical reference. Choosing the reference from finite continuations does not identify the population oracle. Its evaluated gap can be negative at a single state even though the aggregate gaps here are positive.

B.6 Nested Candidate Pools

Table 11: Nested pools with fixed scorer/executor (four-task mean). Base includes no repair and closure. Ref./Sel. denote reference/selected utility; scoring time is per state.

Pool	Distinct actions	Ref.	Sel.	Gap	Score ms
Base	2.0	0.711	0.711	0.000	0.18
+ Target	2.9	0.728	0.725	0.003	0.25
+ Depth-1	3.8	0.751	0.746	0.005	0.34
+ Depth-2	4.7	0.776	0.763	0.013	0.43
+ Global	5.4	0.782	0.765	0.017	0.49

Starting from $\mathcal{C}_0 = \{a_0, a_{\mathrm{cl}}\}$, the successive pools add $a_{\{v\}}$, $a_{R_1(v)}$, $a_{R_2(v)}$, and $a_{W(s)}$, respectively. Scopes follow Appendix C.2; duplicates are removed and the deterministic advancement rule remains fixed. The same trained scorer and executor evaluate all pools on the same states. Independent continuations evaluate each pool's empirically selected reference and deployed action. The reference utility rises from 0.711 to 0.782 in the task-macro results, while selected utility rises to 0.765. These comparisons test pool expansion without changing the scorer. They measure opportunity relative to the largest evaluated pool; they do not enumerate all feasible repair scopes.

Table 12: Nested-pool reference and selected utilities by task. The first pool contains both no repair and closure. The final pool has at most six distinct actions.

	GSM8K		MATH		HumanEval		MBPP	
Pool	Ref.	Sel.	Ref.	Sel.	Ref.	Sel.	Ref.	Sel.
No repair + closure	0.746	0.746	0.681	0.681	0.704	0.704	0.713	0.713
+ Target	0.763	0.760	0.698	0.695	0.721	0.718	0.730	0.727
+ Depth-1	0.785	0.780	0.724	0.719	0.744	0.739	0.751	0.746
+ Depth-2	0.810	0.797	0.749	0.736	0.769	0.756	0.776	0.763
+ Global	0.817	0.800	0.756	0.738	0.774	0.758	0.781	0.765

Table 13: Candidate expansion on 1,000 shared states.

Pool	Max.	Mean	Ref.	Sel.	ms
Base	2	2.0	0.711	0.711	0.18
Current pool	6	5.4	0.778	0.765	0.49
Expanded-12	12	10.7	0.794	0.776	0.86
Expanded-24	24	19.8	0.803	0.784	1.47
Enumeration	–	32.7	0.806	–	–

Max./Mean count feasible actions; Ref./Sel. denote reference/selected utility; ms measures scoring time per state. Base includes no repair and closure. PRICE utility and both gaps use this shared subset.

Table 14: Utility and gaps on the same states.

Quantity	Utility / gap
PRICE utility	0.765
Candidate omission	0.028
Selection gap	0.013
Total gap	0.041

B.7 Empirical Candidate Coverage and Selection Gaps

We evaluate all pools and the deployed controller on the same 1,000 LLaDA repair states from held-out problems. Starting snapshots, remaining budgets, continuation policy, executor, verifier, and cost normalization stay fixed. Enumeration is an offline coverage diagnostic; training and deployment keep the bounded candidate generator. Table 13 separates template caps from the mean number of feasible enumerated actions.

Relative to the baseline pool, the current pool captures $(0.778-0.711)/(0.806-0.711) = 70.5\%$ of the measured enumeration opportunity. The controller captures $(0.765-0.711)/(0.806-0.711) = 56.8\%$; within the bounded pool it captures $(0.765-0.711)/(0.778-0.711) = 80.6\%$ of the available improvement. The empirical total gap splits as $0.806-0.765 = (0.806-0.778)+(0.778-0.765) = 0.028+0.013$ (Table 14), connecting coverage and selection to Proposition 3. These values give an empirical decomposition based on finite continuations. They do not measure the population optimum or the uniform value-error bound. The analysis in Table 4 uses different states; we exclude its 0.017 gap from this decomposition.

The expanded pools add intermediate descendant scopes, scopes spanning multiple branches, and sparse descendant combinations to the original scope family. After removing infeasible and duplicate actions, the mean counts are 10.7 and 19.8 under caps of 12 and 24. The state snapshots, continuation, and executor are shared. Expansion raises selected utility from 0.765 to 0.776 and 0.784, while scoring costs rise from 0.49 to 0.86 and 1.47 ms/state. Expanded-24 captures 96.8% of the measured reference opportunity, whereas its selected policy captures 76.8%. The within-pool selection gap is 0.019, compared with 0.013 for the current pool. Broader coverage and more accurate selection are separate sources of improvement.

B.8 Scorer and Supervision Controls

Table 3 holds the candidate pool fixed while changing the scorer. The probability-additive predictor is a restricted diagnostic. Additive logits provide a nonlinear control (Appendix A.5). The joint configuration performs better, supporting complete-action modeling. This does not prove that a specific encoder is necessary. The outcome-informed oracle uses candidate outcomes; its deployment inference cost is unknown.

The paired comparison matches the encoder, pool, and continuation-call count. Equal call counts need not give equal teacher computation when rollout lengths differ. Paired differences contain no extra information when the same absolute returns are already available. To isolate the effect of target differencing, we need return regression on the same data. To isolate collection efficiency, we need matched teacher cost. The reported 0.4-point gains describe the tested training configurations; they do not guarantee reduced variance.

B.9 Dependency Perturbations

Table 15 reports perturbations of the predicted graph on LLaDA/GSM8K. At 20% edge removal, accuracy is 81.4% and NFE is 35.5; at 20% edge addition, they are 81.7% and 37.6. The unperturbed configuration gives 82.6% and 36.0. The graph controls both quarantine and update eligibility, so these changes can affect candidate availability, visibility, and scheduling together. Unperturbed

Table 15: Predicted-graph perturbations on LLaDA/GSM8K. Unperturbed uses the original predicted graph.

Graph	Acc. (%)	NFE ↓
Unperturbed	82.6	36.0
10% missing	82.1	35.9
20% missing	81.4	35.5
30% missing	80.3	35.2
10% spurious	82.2	36.8
20% spurious	81.7	37.6
30% spurious	80.9	38.5

Table 16: LLaDA cost-weight sweep. Scores are percentages; costs are four-task macro-averages. Main: $\lambda = 0.20$.

λ	GSM8K	MATH	NFE	Rel. lat.
0.00	83.0	47.0	39.4	1.21
0.10	82.8	46.9	37.7	1.17
0.20	82.6	46.8	36.2	1.13
0.40	82.1	46.2	34.8	1.09
0.80	80.9	45.0	33.5	1.05

Table 17: Graph-source controls on LLaDA/GSM8K.

Graph source	Acc. (%)	NFE ↓
Random DAG	80.4	35.4
20% edge rewiring	81.5	35.6
Predicted	82.6	36.0
Trace-derived approx.	83.0	36.1

Table 18: Scorer refresh under deployment shift. Agreement is with the empirical best-in-pool action.

Scorer	State policy	Agree. (%)	Utility
Frozen	Original	72.5	0.765
Frozen	PRICE	65.8	0.741
Refresh 1	PRICE	68.9	0.753
Refresh 2	PRICE	70.6	0.758

does not mean that the graph is semantically correct. To separate these effects, we must change the proposer and executor separately while keeping other controls fixed.

B.10 Graph Source Controls

We vary the graph source with the backbone, repair states, continuation, executor, and scorer fixed (Table 17). The random DAG matches the predicted graph's node and edge statistics. The 20% rewiring condition changes the identities of 20% of the original predicted edges, using the original edge count $|E|$ as the denominator. The total edge count and graph density stay fixed. We extract dependencies from reasoning traces to form an approximate graph. This is an additional graph-source control, not a ground-truth DAG. The predicted graph improves accuracy over the random DAG by 2.2 points; the trace-derived approximation adds 0.4 points. Graph source affects candidate construction, quarantine, and update eligibility together. Table 15 separately reports the edge-removal and edge-addition sweep.

PRICE quarantines potential descendants, including those that remain valid after a change. Dependency coverage therefore depends on reachability, while the need for repair depends on actual values. If a missing path leaves a stored descendant outside $D_s(R)$, it can remain visible without a reuse check. Missing a direct edge need not cause this failure when another path preserves reachability.

B.11 Affected-Region Diagnostics

For each repair state, $D_s(R)$ is the predicted potentially affected region. Matched interventions yield the realized affected-region reference $D_s^\star(R)$. It contains units flagged by changes in unit values or terminal correctness after an upstream intervention. This reference is used only for offline diagnostics; it is not a ground-truth dependency graph. Precision, recall, and F1 compare $D_s(R)$ with $D_s^\star(R)$ (Table 19). The reference measures actual invalidation, not all possible dependencies. A potentially affected descendant may remain valid. Lower precision therefore need not mean a graph error or a need to regenerate that descendant.

Table 20 compares accuracy using the realized reference and predicted affected regions. Both conditions use the same states within each recall group. Differences between groups also reflect the states they contain; they do not identify a causal effect of recall alone. Affected-region recall is distinct from the value-based candidate omission term $\eta_{\mathcal{C}}(s)$. Table 21 compares quarantine costs using regions from the realized reference, the predicted graph, and a random DAG. Additional NFE is

Table 19: Affected-region prediction on LLaDA. All metrics are percentages.

Task	Precision	Recall	F1
GSM8K	87.4	84.5	85.9
MATH	86.1	82.9	84.5
HumanEval	89.2	86.7	87.9
MBPP	88.5	85.8	87.1

Table 20: LLaDA/GSM8K groups by recall. Ref. uses $D_s^\star(R)$; Pred. uses $D_s(R)$. Accuracy is in %; Gap is Ref. minus Pred. in points.

Recall (%)	States	Ref.	Pred.	Gap
> 95	260	83.0	82.8	0.2
$[90, 95]$	410	82.7	82.1	0.6
$[80, 90)$	250	82.4	81.0	1.4
< 80	80	81.8	78.7	3.1

measured relative to the condition using $D_s^\star(R)$. These comparisons are offline diagnostics; $D_s^\star(R)$ is unavailable to PRICE during deployment. The trace-derived approximation in Table 17 is a separate graph-source control.

Table 21: Offline quarantine diagnostics on LLaDA/GSM8K. Units and NFE are means; additional NFE is relative to the realized affected-region reference.

Region source	Quarantined units	Additional NFE	Accuracy (%)
Reference region	3.1	0.0	82.9
Predicted	3.6	+0.7	82.6
Random	5.5	+2.2	80.4

B.12 Policy Shift and Scorer Refresh

The initial scorer is trained on states from Original trajectories on training problems. Refresh follows the same problem-level split (Appendix C.5). For evaluated states $\mathcal{S}_{\text{eval}}$ with count $n_{\text{states}} = |\mathcal{S}_{\text{eval}}|$, top-1 agreement is $\frac{100}{n_{\text{states}}} \sum_{s \in \mathcal{S}_{\text{eval}}} \mathbf{1}[\hat{a}(s) = \tilde{a}_{\mathcal{C}}(s)]$. Independent continuations estimate the empirical best action $\tilde{a}_{\mathcal{C}}$. The frozen scorer reaches 72.5% agreement and 0.765 utility on Original states, versus 65.8% and 0.741 on PRICE states (Table 18). Each refresh collects deployment states and retrains the scorer. Two rounds partially recover agreement and utility to 70.6% and 0.758. Utility uses normalized remaining NFE with $\lambda = 0.20$ (Appendix B.4). These online metrics evaluate refreshed controllers. State collection and retraining have separate offline costs. Across state distributions, utility differences reflect both repair opportunity and selection quality. Within-pool regret on a fixed evaluation set more directly measures selection quality. Top-1 agreement measures action selection rather than reward/cost calibration.

B.13 Offline Intervention and Refresh Cost

Table 22: Offline collection and scorer training on one RTX 5090 (32 GB). Each refresh row reports only that round's additional work.

Stage	States	Rollouts	Rollout GPU-hours	Scorer training GPU-hours	Total offline GPU-hours
Initial	120,000	1,350,000	224.6	6.4	231.0
Refresh 1	30,000	320,000	53.1	1.8	54.9
Refresh 2	30,000	330,000	55.0	1.9	56.9

Table 22 reports state and continuation counts and offline computation on one GPU. Rollout collection includes planning; scorer training is reported separately. Initial collection and training total 231.0 GPU-hours. Refresh 1 and Refresh 2 add 54.9 and 56.9 GPU-hours, respectively, bringing the three-stage total to 342.8 GPU-hours. Offline data collection and scorer optimization are separate from online inference and excluded from the reported online NFE and latency.

Table 23: Verification counts and reference-consistency rates. $A_{\neq}$ counts accepted reference disagreements; $R_{=}$ counts rejected reference agreements. $r_{=} = R_{=}/(A - A_{\neq} + R_{=})$ conditions on reference agreement, not semantic correctness.

	Event counts					Reference-consistency rates (%)			
Task	N	A	$N-A$	$A_{\neq}$	$R_{=}$	A/N	$A_{\neq}/N$	$A_{\neq}/A$	$r_{=}$
GSM8K	4,286	2,734	1,552	77	142	63.8	1.8	2.8	5.1
MATH	14,732	8,750	5,982	354	617	59.4	2.4	4.0	6.8
HumanEval	612	405	207	10	32	66.2	1.6	2.5	7.5
MBPP	1,845	1,194	651	35	79	64.7	1.9	2.9	6.4

Table 24: Paired terminal-outcome audit of accepted reuse. W counts reuse failures with successful reference regeneration; B counts both-failed cases.

Task	Attempts N	Accepted A	Harmful W	Both failed B	W/A (%)	W/N (%)
GSM8K	4,286	2,734	36	18	1.32	0.84
MATH	14,732	8,750	257	41	2.94	1.74
HumanEval	612	405	8	3	1.98	1.31
MBPP	1,845	1,194	26	6	2.18	1.41

B.14 Reward–Cost Operating Points

The sweep in Table 16 shows quality–cost operating points. Increasing λ from 0 to 0.80 reduces NFE from 39.4 to 33.5 and relative latency from 1.21 to 1.05; GSM8K decreases from 83.0% to 80.9% and MATH from 47.0% to 45.0%. Changing the deployment penalty can also change the states visited later and continuation values. Separate reward and cost heads support rescoring, but they do not by themselves establish calibrated values or optimal policies at every penalty. Comparisons across operating points must use the same cost definition and normalization.

B.15 Execution Controls and Verification Accounting

The executor comparison in Table 6 holds the decision state, selector, selected R, continuation, remaining budget, and seeds fixed. Visible-stale repair leaves unselected affected descendants visible. Full regeneration clears and regenerates all of $D(R)$. Selective retention clears the same region, discards R, and permits verification-gated reuse outside R; rejected values are regenerated. Permanent discard and quarantine without reuse are the same when the shared executor eventually regenerates every affected unit, so they form one baseline. The results compare these complete execution configurations; they do not isolate the effect of visibility alone.

Reference-consistency diagnostics. Each probe fully masks the cached slot, hiding its content from the backbone. It accepts only exact content agreement with a valid format. Table 23 describes the checks. For this offline diagnostic, cached values are compared with outputs from paired full-regeneration branches under the same state and continuation. These references are unavailable to the online verifier. Let N, A denote attempt and acceptance counts, $A_{\neq}$ accepted reference disagreements, and $R_{=}$ rejected reference agreements. The reported rates are A/N, $A_{\neq}/N$, $A_{\neq}/A$, and $R_{=}/(A - A_{\neq} + R_{=})$. For example, GSM8K gives $2734/4286 = 63.8\%$ acceptance and $77/2734 = 2.8\%$ disagreement among accepted values. Agreement with a stochastic regeneration branch provides a consistency reference, not proof of semantic correctness. These rates do not estimate the semantic false-acceptance probability α in Proposition 1.

For acceptance event A and an independently assessed semantic-error event E, the theoretical quantity is $\alpha(s) = \Pr(A \cap E \mid s) = p(s)\Pr(E \mid A, s)$. Thus incorrect acceptances per attempt and errors among accepted values have different denominators. A semantic label concerns the cached unit under its current inputs; terminal-answer correctness measures downstream performance and does not by itself label an intermediate cached value.

Table 25: Verification strategies on LLaDA/GSM8K.

Strategy	Acc.	Reuse	NFE	Lat.
Full regen.	80.4	0.0	40.9	1.27
Consistency	82.6	63.8	36.0	1.12
Task-aware	82.8	55.4	37.2	1.17

Table 26: Local verification cost (NFE/check).

Task	Probe	Reject	Reuse	Regen	Save
GSM8K	1.00	1.06	2.06	2.94	0.88
MATH	1.00	1.45	2.45	3.58	1.13
HE	1.00	1.09	2.09	3.21	1.12
MBPP	1.00	1.09	2.09	3.08	0.99

Acc. and Reuse are percentages; Reuse is the check acceptance rate, and Lat. is relative to Original within this comparison. Local cost: Reject is additional rejection continuation; Regen is direct regeneration; Save is net saving.

Paired terminal outcomes. Table 24 compares final-task outcomes following accepted reuse with paired reference regeneration. Harmful reuse means that the reuse continuation fails while the reference succeeds. Cases where both fail are reported separately. W/A uses accepted checks, while W/N uses all attempted checks. The remaining $A - W$ cases include both-failed outcomes; they do not establish the semantic correctness of individual cached units.

For binary terminal rewards, W counts observed positive reward-loss events after accepted reuse under the paired continuations. This tests the reward-loss component of Proposition 1, separately from the semantic-error event used in its sufficient condition.

Task-aware verification. The task-aware checker applies symbolic/numerical equivalence to mathematical results using cached values, model predictions, and current parents. Code checks execute in the public environment with public tests; other units use consistency, parsing, type, and dependency-version checks. Neither verifier accesses benchmark reference answers or hidden tests during inference. Relative to the default probe, the stronger checker gains 0.2 accuracy points at 1.2 additional NFE and a latency ratio of 1.17 versus 1.12 (Table 25). Its acceptance rate is 8.4 points lower. These operating points show the trade-off between additional checking and recomputation; higher final accuracy alone does not identify a lower per-unit false-acceptance rate.

Local and complete cost accounting. Each check costs one NFE. Rejection continuation is averaged over all attempts, with accepted checks contributing zero; the reuse path adds this cost to the probe. Direct regeneration starts from the same cleared state (Table 26). Shared forwards count once; end-to-end latency also includes all non-forward overhead. Let G_{full} be the realized NFE for direct regeneration and $G_{\text{after reject}}$ the additional regeneration NFE after rejection. The measured local saving is

$$\mathbb{E}\big[G_{\text{full}} - \big(1 + \mathbf{1}[\text{reject}]G_{\text{after reject}}\big)\big] .$$

The latter cost excludes the already charged probe. The per-attempt savings in Table 26 differ from the complete action's remaining-cost difference. Let $N_{\text{full}}(s)$ and $N_{\text{selective}}(s)$ be random remaining forward counts under full regeneration and selective retention, respectively. Their expected difference is

$$\Delta N_{\text{reuse}}(s) = \mathbb{E}[N_{\text{full}}(s) - N_{\text{selective}}(s)] , \tag{18}$$

which counts every later backbone call through termination or budget exhaustion, including changed downstream work. Dividing this difference by $\kappa(s)$ gives the normalized NFE cost difference in the action-value objective.

B.16 Task-Specific Computation and Latency

Table 27 gives the task-level values used in Table 1. For example, LLaDA PRICE uses 36.0 NFE on GSM8K and 37.3 on MATH, giving 36.7 after rounding their two-task mean; its four-task mean is 36.2. The full-closure values are 40.9 on GSM8K, 42.1 on MATH, 41.5 for those two tasks, and 41.3 for all four. The cost-weight sweep reports the four-task macro NFE and latency.

Figure 2 compares native-termination operating points using task-specific costs. Relative to closure, PRICE gains 2.2/2.3 accuracy points on GSM8K/MATH while reducing their latency ratios by approximately 11.8%/12.2%. Compared with Confidence, latency differences are small and quality gains are larger. The cost-weight sweep in Table 16 reports macro costs and is not mixed with these task-level axes.

Table 27: Task-level NFE / relative latency. Latency is normalized to Original on the same backbone; Macro averages the four tasks equally.

Backbone	Method	GSM8K	MATH	HumanEval	MBPP	Macro
LLaDA	Original	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00
	Target-only-visible	34.0 / 1.05	34.6 / 1.07	33.8 / 1.05	34.0 / 1.07	34.1 / 1.06
	Full Closure	40.9 / 1.27	42.1 / 1.31	41.0 / 1.26	41.2 / 1.28	41.3 / 1.28
	Confidence	36.7 / 1.13	37.6 / 1.16	36.2 / 1.13	36.7 / 1.14	36.8 / 1.14
	PRICE	36.0 / 1.12	37.3 / 1.15	35.5 / 1.12	36.0 / 1.13	36.2 / 1.13
iLLaDA	Original	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00
	Target-only-visible	33.7 / 1.04	34.2 / 1.06	33.5 / 1.05	33.8 / 1.05	33.8 / 1.05
	Full Closure	40.3 / 1.26	41.5 / 1.29	40.5 / 1.27	40.5 / 1.26	40.7 / 1.27
	Confidence	36.0 / 1.12	36.8 / 1.15	35.7 / 1.12	35.9 / 1.13	36.1 / 1.13
	PRICE	35.5 / 1.11	36.3 / 1.14	35.3 / 1.11	35.7 / 1.12	35.7 / 1.12
Dream	Original	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00	32.0 / 1.00
	Target-only-visible	33.9 / 1.05	34.6 / 1.07	33.7 / 1.06	33.8 / 1.06	34.0 / 1.06
	Full Closure	40.6 / 1.26	41.8 / 1.29	40.5 / 1.26	40.7 / 1.27	40.9 / 1.27
	Confidence	36.4 / 1.13	37.2 / 1.16	36.2 / 1.13	36.2 / 1.14	36.5 / 1.14
	PRICE	35.9 / 1.12	36.7 / 1.15	35.5 / 1.12	35.9 / 1.13	36.0 / 1.13

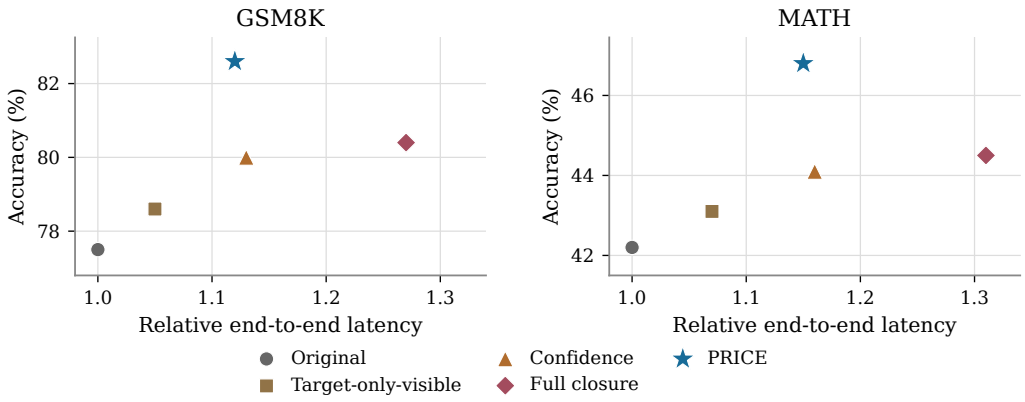


Figure 2: Task-specific accuracy–latency operating points for LLaDA-8B-Instruct. Scores use the three-seed means in Table 1; latency uses the task-level measurements in Table 27, normalized to Original separately for each task. PRICE uses $\lambda = 0.20$. Each marker denotes one reported configuration.

C Implementation and Control Details

The backbone checkpoints are `GSAI-ML/LLaDA-8B-Instruct`, `GSAI-ML/iLLaDA-8B-Instruct`, and `Dream-org/Dream-v0-Instruct-7B` (Nie et al., 2025; 2026; Ye et al., 2025).

C.1 Planning and state validity

Planner and training. Each backbone serves as its own planner: LLaDA-8B-Instruct, iLLaDA-8B-Instruct, and Dream-v0-Instruct-7B use their respective frozen checkpoints listed above. Structured prompting uses their existing instruction-following ability. Planning requires no separate model, extra planner training, or graph annotations. The PRICE scorer is trained separately on offline intervention outcomes.

Graph generation. Planning runs once before reasoning and takes only the question q as input. The backbone generates reasoning units, their descriptions and result types, dependency edges, and fixed token-slot assignments, including a final-answer unit. Reference solutions are unavailable to

the planner. We parse the structured output, validate its schema, and check for cycles. The resulting DAG stays fixed within the episode. It guides the deterministic candidate constructor, affected-region estimation, and update eligibility. The scorer estimates candidate reward–cost values and selects the action; the planner supplies neither action-value predictions nor reward targets. Invalid plans, slot overflows, and parser failures use the predefined budgeted fallback and remain in the evaluation denominator. A resolved unit must have a parsed terminator, the required type, no required masks, and references to current parent versions. These conditions make the state valid for execution but do not ensure semantic correctness.

Planning cost. A planning generation uses roughly 80–120 output tokens on average. Table 28 reports its approximate per-episode cost on one RTX 5090 with 32 GB memory. Reusing the backbone still incurs planning forwards. These are already included in reported NFE and end-to-end latency, and count toward the 40-NFE cap for methods that invoke planning. Planning is charged once per episode. It precedes the cloned repair states and is not charged again in each candidate's remaining-cost target. Offline collection costs in Table 22 also include planning.

Table 28: Approximate planning cost per episode on one RTX 5090 (32 GB). Planning runs once and is included in reported online costs.

Backbone	Latency (ms)	NFE
LLaDA-8B-Instruct	≈ 120	≈ 2
iLLaDA-8B-Instruct	≈ 110	≈ 2
Dream-v0-Instruct-7B	≈ 130	≈ 2–3

C.2 Deterministic candidate construction

The depth-based candidate construction uses the shortest directed path distance $d_{\widehat{G}}(v, u)$ and the ordered scope list

$$\varnothing, \quad \{v\}, \quad R_1(v), \quad R_2(v), \quad D_s(\{v\}), \quad W(s),$$
$$R_k(v) = \{v\} \cup \{u \in \mathrm{Desc}_{\widehat{G}}(v) \cap W(s) : d_{\widehat{G}}(v, u) \le k\}.$$

Among units with stored content that have not reached their repair cap, choose the unit with the lowest recorded confidence as the target v, breaking ties by unit identifier. Exclude scopes that would exceed a unit's repair cap. Keep feasible scopes in the order above and remove duplicates by their sorted identifier sets. The no-repair action remains available even if no target is eligible. Equal predicted action values are resolved by this fixed pool order. The pool has at most six actions, each with $U = g(s, R)$. Empty repair means ordinary advancement, including STOP where applicable. These scopes have a distance limit but need not include all ancestors. Nested-pool comparisons restrict this construction while keeping the same target selector, feasibility rules, and tie-breaking. The enumeration check in Appendix B.7 does not change this deployment pool.

C.3 Candidate controls and online information

Candidate ranking, feasibility, tie-breaking, and the pool cap are fixed before collection, with duplicates removed. The shared deterministic advancement rule maps cleared states to eligible updates. Scope comparisons keep this rule but allow different frontiers. Scorer comparisons use the same pool. Online features use observations made before the action. Extra masked probes are shared and charged; features from future candidate rollouts are unavailable.

Scope selection runs at the initial state, after a fixed number of completed control intervals, and at formal completion. Between these checkpoints, the executor follows ordinary advancement without ranking repair scopes again. Each interval fixes its eligible set U, checks cached values, and advances generation for eligible unresolved units. Newly eligible units enter a later interval. An interval can require several forward passes, so interval counts are distinct from NFE. All candidate features at a checkpoint use the same pre-action state; candidate membership flags describe a proposed action, not its observed outcome.

C.4 Verification protocol

Equation 7 defines the one-pass consistency probe. Equality covers every token to be restored, including its termination structure, under the fixed slot serialization; padding outside restored content is ignored. We use greedy decoding: each position takes the argmax with fixed tie-breaking. The cache is read only by the external comparator after prediction. This probe uses one pass, rather than a full diffusion regeneration trajectory. An accepted value is restored with current parent versions. A rejected value is discarded and its slot remains masked. Probe predictions do not initialize regeneration. The executor makes a fresh generation call under the shared continuation rule, using the remaining budget. Each attempted check costs one NFE. Every later forward pass is charged separately, and latency includes all elapsed time. Agreement does not ensure semantic correctness.

C.5 Matched intervention supervision

Original problems are assigned to training, development, and test splits before states are generated. All states from one problem stay in its assigned split. Evaluation test problems are excluded from training and hyperparameter selection. Controller settings are selected on development data and fixed before test evaluation. Branches share the starting snapshot, remaining budget, continuation policy, executor, advancement rule, and reuse test. For corresponding events, random draws are paired by problem, slot, position, and local update count. Using the same seed alone does not establish pairing. Reference answers score outcomes but never provide replacement intermediate values. Failed or truncated continuations keep their predefined outcomes and costs.

Paired differences add no information when the same absolute returns are available. To isolate the effect of pairing, comparisons must match available features, encoder capacity, candidate pool, and teacher computation. When the generator or controller changes, states and continuation labels are collected again under the updated policy. Separate reward and cost heads allow rescoring but do not ensure calibration for a new continuation policy.

C.6 Progress and termination

Generation and verification consume the remaining computation budget. Actions must advance computation, remove content, or stop. A fixed progress fallback prevents confidence thresholds from stalling generation. Per-unit and episode repair caps limit reopening. Formal completion leads to a budgeted STOP-versus-repair checkpoint. When the budget runs out, the system returns the current parsed answer or records a failure. If planning or slot handling fails before a final answer is available, the executor falls back to the backbone's native generation procedure on the task prompt. This fallback uses only the remaining episode budget; it does not reset the budget or erase earlier planning, generation, or verification costs. If the remaining budget cannot support generation, the episode records a failure and stays in the evaluation denominator. Offline continuations use the same terminal procedure before reward and remaining cost are recorded. For cost labels, the supplementary reference code uses the larger of one NFE and the estimated mean no-repair cost as its shared scale. This keeps the scale positive for zero-cost STOP baselines.

D Notation Reference

Table 29 summarizes the main symbols used in the method, theory, and experiments. The arguments π, b, and λ are omitted from $P(s,a)$, $C(s,a)$, and $Q(s,a)$ when fixed. The table distinguishes sets, population values, learned predictions, and empirical diagnostics.

Table 29: Notation reference. Each group follows the order in which its symbols first appear in the paper.

Symbol	Meaning	Defined in
State and dependency graph		
s	Partial generation state $(q, x, m, \widehat{G}, I, \nu, H, b)$.	Sec. 3.1
q; x, m	Input question; visible token canvas and mask configuration.	Sec. 3.1

Continued on the next page

Table 29: Notation reference (continued).

Symbol	Meaning	Defined in
$\widehat{G} = (V, E)$	Predicted dependency DAG; V and E are its unit and edge sets. V here is a set, distinct from the value function $V(s, a)$.	Sec. 3.1
u, v; $\mathrm{Pa}_{\widehat{G}}$, $\mathrm{Desc}_{\widehat{G}}$	Reasoning-unit indices (with v the selected target); parents and descendants in the predicted graph.	Sec. 3.1
I; ν	Resolved-unit set; version and discard metadata. Resolution requires valid structure and current parents, not a correct task answer.	Sec. 3.1
H; b	External stored-text cache, separate from model KV caches; remaining computation budgets.	Sec. 3.1
$F_{\widehat{G}}(I)$	Eligible frontier: unresolved units whose predicted parents are all resolved.	Sec. 3.1
$W_{\mathrm{vis}}(s)$, $W_{\mathrm{cache}}(s)$; $W(s)$	Units with visible generated text, units with text stored only in the cache, and their disjoint union.	Sec. 3.1
Repair scopes and executable actions		
R	Selected stored-unit scope to discard and regenerate.	Eq. 1
$D_s(R)$; $D(R)$	Stored units in R or its predicted descendants; $D(R)$ is short for $D_s(R)$. This is the affected region, not the chosen scope.	Eq. 1
$D_{\mathrm{vis}}(R)$, $D_{\mathrm{cache}}(R)$	Visible and already cached parts of $D_s(R)$; only the visible part must be removed from the canvas.	Sec. 3.1
$T_R^-(s)$; I^-	Cleared state after invalidation and cache handling; remaining resolved units $I \setminus D_s(R)$.	Eq. 2
g_0; $g(s, R)$; U	Fixed advancement rule, its application to the cleared state, and the resulting eligible update set: $U = g(s, R) = g_0(T_R^-(s))$.	Eq. 2
$a_R = (R, U)$; a_0	Complete action defined by R; no-repair action, which advances normally or stops at a terminal checkpoint.	Secs. 3.1–3.2
$\mathcal{A}(s)$; $\mathcal{C}(s)$	Feasible induced-action space and bounded candidate pool. The calligraphic $\mathcal{C}$ is a set, not computation cost.	Secs. 3.1–3.2
$R_k(v)$; $d_{\widehat{G}}(v, u)$	Target plus stored descendants within depth k; shortest directed path distance used to define that depth.	App. C.2
a_{cl}; a_S	Full predicted-closure action and a selective action that retains a descendant for checking in Proposition 1.	Secs. 4.1–4.2
Reward, cost, and learned action values		
π; Y; $r(Y)$	Fixed continuation policy, terminal output, and its task reward in $[0, 1]$.	Eq. 3
$P_b^\pi(s, a)$	Expected terminal reward after executing a and continuing under π with budget b; abbreviated $P(s, a)$.	Eq. 3
C_{raw}; $\mu_C(s, a)$	Actual raw remaining cost and its expectation; includes verification and work later discarded.	Sec. 3.1
$\kappa(s)$; $C_b^\pi(s, a)$	Positive cost scale shared by all candidates; normalized expected cost $\mu_C(s, a)/\kappa(s)$, abbreviated $C(s, a)$.	Eq. 3
λ; $Q_{\lambda,b}^\pi(s, a)$	Nonnegative cost weight; expected reward–cost value $P(s, a) - \lambda C(s, a)$, abbreviated $Q(s, a)$.	Eq. 3
$\mathcal{I}_s^Z(A, B)$	Interaction of disjoint repair regions A, B for $Z \in \{P, C, Q\}$, comparing their joint effect with their separate effects.	Eq. 4
Δr_a, Δc_a	Expected reward and normalized-cost differences from the no-repair action a_0.	Sec. 3.3
$V(s, a)$; $\widehat{V}(s, a)$	True value relative to the baseline, $Q(s, a) - Q(s, a_0)$; its learned estimate $\widehat{\Delta r}_a - \lambda \widehat{\Delta c}_a$.	Eq. 5
$\widehat{\Delta r}_a$, $\widehat{\Delta c}_a$; $\overline{\Delta r}_a$, $\overline{\Delta c}_a$	Reward/cost head predictions; their targets estimated from matched offline continuations.	Secs. 3.3–3.4
$\mathcal{L}$; β; c_0	Scorer training loss, cost-loss weight, and training-only cost residual scale. c_0 is separate from $\kappa(s)$.	Sec. 3.4
$\tilde{a}$; $\hat{a}$; δ	Predicted best candidate, executed action, and deployment margin; gains of at most δ trigger a_0.	Eq. 8
Verification and theoretical bounds		
h; J_h; c_h	Cached unit being checked, its token slot, and its stored token sequence. Here h is a unit index.	Sec. 3.5

Continued on the next page

Table 29: Notation reference (continued).

Symbol	Meaning	Defined in
x^-; $\widetilde{c}_h$	Cleared model input and token sequence predicted by the probe; the cached sequence is absent from the input.	Eq. 7
p_θ; θ; d_{check}; $\text{Verify}(h)$	Token distribution, frozen parameters, probe decoder, and acceptance indicator. ParseOK checks output format.	Eq. 7
g_C, g_F; v, h	Normalized regeneration cost under closure and after rejection; verification cost and extra overhead. Here v, h are costs, not unit indices.	Prop. 1
C_{base}; g	Shared other-work cost and common regeneration cost when $g_C = g_F = g$, in normalized units.	App. A.2
p; α; L	Acceptance probability; probability of accepting incorrect reuse per attempt; bound on terminal reward loss on that event.	Prop. 1
$a^*(s)$; $a^*_{\mathcal{C}}(s)$	Best true-value action in the feasible space and in the candidate pool, respectively, under the fixed continuation.	Eq. 9
ϵ; ϵ_r, ϵ_c	Uniform relative-value error and error bounds for each head, with $\epsilon = \epsilon_r + \lambda\epsilon_c$.	Prop. 2
$\Delta_{\text{cl}}(s)$	True-value advantage of the best candidate over full closure.	Eq. 12
$\eta_{\mathcal{C}}(s)$	Candidate omission: true-value gap between the best feasible action and the best candidate.	Eq. 13
ξ; K_θ	Visible input/control metadata and replacement kernel at a fixed call, used to analyze isolation from quarantined stored text.	Prop. 4
$F_s(R)$; $\widehat{F}_s(R)$	Scope reward and its probability-additive approximation; distinct from frontier $F_{\widehat{G}}$.	App. A.5
Empirical utility and verification diagnostics		
$\bar{C}(s,a)$; $U_\lambda(s,a)$	Baseline-normalized NFE change; utility $P - \lambda\bar{C} = Q + \lambda$ when $\kappa(s) = \mu_C(s,a_0) > 0$; distinct from update set U.	Eq. 17
$\tilde{a}_{\mathcal{C}}$; u_a	Empirical pool-reference action selected on one continuation sample; action utility estimated on disjoint evaluation samples.	Sec. 5
$\overline{U}_{\text{sel}}(s,a)$	Utility estimated from the continuations used to select the empirical pool reference.	App. B.5
$\mathcal{S}_{\text{eval}}$; n_{states}	Evaluated repair states and their count for top-1 action agreement; distinct from verification-attempt count N.	App. B.12
$\widehat{\Delta}_{\text{opp}}$, $\widehat{\Delta}_{\text{PRICE}}$; $\widehat{G}_{\text{ref}}$	Empirical reference and PRICE utility gains over closure; their difference, the empirical within-pool selection gap.	Eq. 14
$D^\star_s(R)$	Realized affected-region reference from matched interventions; used only for offline diagnostics, not a ground-truth dependency graph.	App. B.11
N, A, W, B	Attempts, accepts, reuse-failed/reference-succeeded outcomes, and both-failed outcomes; W/A conditions on acceptance, W/N on all attempts.	Table 24
$A_{\neq}$; $R_{=}$	Accepted disagreements and rejected agreements with the paired regeneration reference. These measure reference consistency, not unit correctness.	Table 23
$r_{=}$	Rejection rate among reference-agreeing checks: $R_{=}/(A - A_{\neq} + R_{=})$; denominator differs from N and A.	Table 23
$\Delta N_{\text{reuse}}(s)$	Expected total remaining-NFE saving from selective execution versus full regeneration, including later work.	Eq. 18
$\overline{\text{NFE}}$; $\overline{L}_{\text{rel}}$	Mean forward count across four tasks and mean of task-specific latency ratios to Original; each task is weighted equally.	Sec. 5