LEARNING A REUSABLE TOKEN-COMMITMENT POLICY FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models refine many token positions in parallel, but efficient decoding requires deciding when to commit proposed tokens. We ask whether token commitment can be learned as a reusable policy over diffusion traces, rather than calibrated separately for each decoding configuration. We introduce TRACELOCK, a lightweight controller for a frozen diffusion backbone. An intermediate token proposal is labeled future-stable when it matches the final token of its completed trace; a shared contextual scorer learns from these labels on variable-length trace states. A single TRACELOCK checkpoint per backbone is deployed across the tested window widths and generation lengths without collecting new traces or retraining. Experiments on mathematical reasoning, question answering, and code generation show an improved quality–step tradeoff over heuristic and learned baselines and greater stability than Learn2PD when the decoding window changes. We use one global acceptance cutoff across tasks and configurations, avoiding per-setting calibration; changing that cutoff provides a controllable quality–step tradeoff. These results support treating token commitment as a reusable learned policy over evolving diffusion traces.

1 INTRODUCTION

Diffusion large language models (D-LLMs) have emerged as a promising alternative to autoregressive large language models (AR-LLMs), especially in masked discrete diffusion settings (Sahoo et al., 2024; Ou et al., 2025) such as LLaDA and Dream (Nie et al., 2025; Ye et al., 2025). Unlike AR-LLMs, which irreversibly append tokens from left to right, masked D-LLMs iteratively refine a partially masked sequence and can update multiple token positions in parallel. This parallel refinement avoids the fixed generation order of autoregressive decoding and creates the possibility of faster generation. In practice, however, high-quality D-LLM generation often still requires many denoising iterations, each involving bidirectional attention over the current sequence. Fast decoding is therefore a central challenge for deploying D-LLMs effectively (Wu et al., 2025b; Chen et al., 2025b; Israel et al., 2025).

Fast decoding for D-LLMs is often described as reducing the number or cost of denoising steps. In the frozen-generator setting, however, the generator is fixed and the main algorithmic freedom lies elsewhere: the decoder must decide which proposed tokens should stop being revised. *This turns fast decoding into a token-commitment problem.* At each denoising step, the frozen model proposes token values for multiple positions, while the decoder must decide which positions should be committed and which should remain revisable. Choosing an effective commitment policy is non-trivial: committing too late limits speedup, while committing too early can lock in errors. Figure 1 illustrates this tradeoff. Existing frozen-generator decoders can be understood as setting-specific ways of calibrating commitment. Training-free methods use hand-designed confidence thresholds, transfer rules, or block schedules (Wu et al., 2025b;a; Dong et al., 2025), while existing learned filters can tie the acceptance rule to a particular decoding interface, as in Learn2PD's block-level formulation (Bao et al., 2026). This raises a natural question: *can token commitment instead be learned as a reusable policy over the evolving diffusion trace, rather than calibrated separately for each decoding configuration*?

Learning such a policy is difficult because the optimal commitment decisions are not directly observable. A completed generation tells us the final sequence, but not when each token should have stopped being revised. We therefore turn commitment learning into a self-supervised trace-prediction problem derived from completed diffusion traces. An intermediate token proposal is labeled as

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

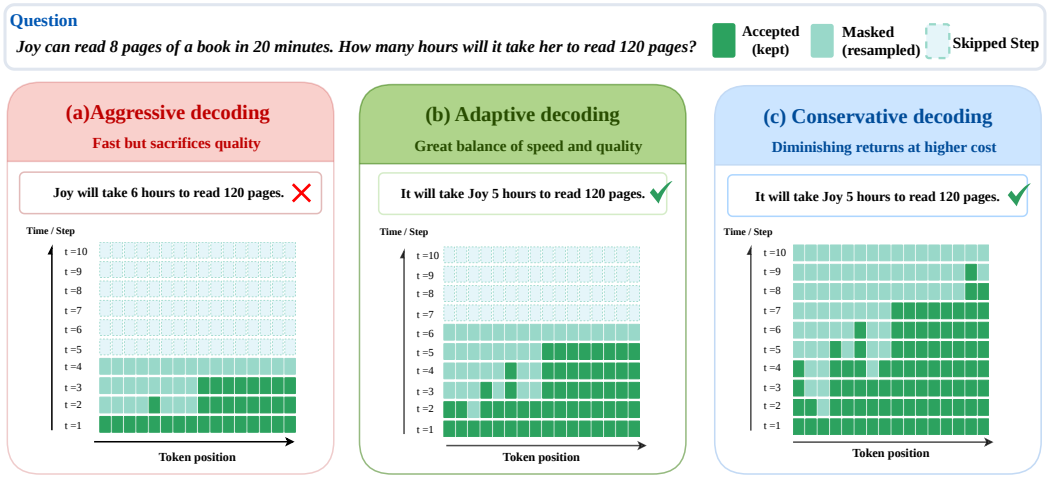


Figure 1: Token commitment as a trace-state decision problem in masked diffusion decoding. Aggressive decoding can be fast but may lock an incorrect trajectory, while conservative decoding keeps tokens revisable for longer at higher cost. Effective commitment therefore depends on the evolving partial trace and can vary across samples and decoding steps.

future-stable if it matches the corresponding token in the final completed sequence. This future-stability target requires no human annotation or task-level correctness label. Although it is only a trace-relative signal, it provides a dense proxy for the online question of which active tokens are stable enough to commit. We propose TRACELOCK, a lightweight plug-in controller that learns a reusable commitment policy from these labels. The base D-LLM remains frozen and TRACELOCK only decides which active generation positions should remain revisable and which should become locked. Rather than designing or learning a setting-specific acceptance calibration, TRACELOCK scores variable-length trace states using frozen-model hidden states, short-range hidden-state dynamics, and prompt/active/locked context, with a single shared contextual scorer. We evaluate TRACELOCK on question answering, mathematical reasoning, and code generation tasks under multiple generation lengths and local-window regimes. The results show that trace-supervised contextual commitment improves the quality–step tradeoff over heuristic and learned decoding baselines in several settings, while the same controller remains stable across changes in local-window width and generation length without retraining. Controlled single-token interventions further show that TRACELOCK-selected commitments agree with the reference continuation about 90% of the time and rarely harm task correctness.

Contributions. We make the following contributions: **Problem formulation.** We formulate efficient frozen-generator D-LLM decoding as a token-commitment problem, where the decoder decides when proposed tokens should become irreversible. **Algorithm.** We show that this commitment policy can be learned from completed diffusion traces using self-supervised future-stability labels, and instantiate it as TRACELOCK, an end-to-end learned commitment controller rather than a hand-designed or block-specific acceptance rule. **Empirical evidence.** Across mathematical reasoning, question answering, and code generation, TRACELOCK improves the quality–step tradeoff over heuristic and learned baselines. One checkpoint per frozen backbone remains stable across the tested window widths and generation lengths without new traces or retraining, and one global acceptance cutoff is used without per-setting calibration. Diagnostics distinguish its commitment decisions from scalar confidence filtering.

2 RELATED WORK

D-LLM decoding acceleration. Discrete and masked diffusion language models have become an increasingly important alternative to autoregressive language modeling (Sahoo et al., 2024; Ou et al., 2025; Nie et al., 2025; Ye et al., 2025). Existing acceleration methods intervene at different levels of the generation pipeline. Systems methods reduce the cost of individual denoising iterations, for example through caching or execution optimizations (Wu et al., 2025b; Jiang et al., 2025). Model-

adaptation methods modify the generator or its training objective so that faster or more aggressive parallel decoding becomes reliable, for example through finetuning, distillation, approximate joint sampling, or learned parallel decoding behavior (Chen et al., 2025b; Bansal & Sanghavi, 2025; Israel et al., 2025). Other methods change the generation process itself through hybrid, blockwise, or forcing-style formulations (Wang et al., 2025b; Arriola et al., 2025). These directions are complementary to our work: TRACELOCK does not train a new diffusion backbone or reduce the cost of an individual denoising step, but learns a reusable inference-time commitment policy over the evolving trace state of a frozen generator.

Frozen-generator decoding acceleration. Our work is closest to methods that keep the D-LLM frozen and accelerate decoding by changing the token acceptance, remasking, or transfer policy. Training-free decoders such as Fast-dLLM and its follow-up variants use confidence-thresholded transfer, blockwise schedules, or related hand-designed rules to decide which positions should be revealed or finalized (Wu et al., 2025b;a). A broader line of work studies alternative heuristics for token ordering, confidence calibration, temporal modeling, or remasking behavior (Li et al., 2025; Wang et al., 2025a; Kim et al., 2025; Hong et al., 2025; Dong et al., 2025). These methods demonstrate that the decoding policy strongly affects the quality–efficiency tradeoff, but their acceptance boundaries are typically hand-designed and may require regime-specific calibration. Recently, Learn2PD (Bao et al., 2026) proposed a method that keeps the base D-LLM frozen and uses agreement with the final decoded sequence to train a lightweight token filter. TRACELOCK shares the idea of learning from future agreement, but differs in the policy being learned. The distinction is therefore not the supervision signal itself, but whether that signal trains a block-specific acceptance filter or a reusable policy over variable-length trace states. Rather than learning a fixed block-level filter tied to a particular decoding interface, TRACELOCK learns a variable-length trace-state commitment policy. Our policy uses contextual hidden states, short-range state dynamics, and a sequence-conditioned threshold, allowing the same checkpoint to be deployed across the tested local windows, generation lengths, and step budgets without changing the architecture or retraining the controller.

3 METHOD

We describe TRACELOCK as a plug-in controller that implements a learned token-commitment policy inside the decoding loop of a frozen D-LLM. The base D-LLM proposes tokens and hidden states; TRACELOCK decides which proposed tokens should become final. At the control level, the goal is to learn a policy that maps the current trace state to commit-or-revise decisions. In the training process, completed generation traces provide self-supervised future-stability labels. For deployment, the same learned controller applies this policy to the current trace state without retraining or per-setting calibration. Figure 2(a) shows how completed traces define the future-stability labels used for training. Figure 2(b) shows the corresponding online decision at deployment: given the current partial trace, the controller predicts which active tokens are stable enough to lock.

3.1 PROBLEM FORMULATION

Let a prompt occupy positions $1, \ldots, P$ and let the generation region have length N, so the total sequence length is $L = P + N$. A masked diffusion language model iteratively updates a sequence

$$x_t \in \mathcal{V}^L, \qquad t = 0, \ldots, T,$$

where T is the total number of steps, x_t are the predicted tokens at step t and $\mathcal{V}$ is the vocabulary. Unfilled generation positions contain a special mask token.

At each step, the frozen D-LLM with parameters ψ produces token logits and hidden states

$$(\ell_t, H_t) = F_\psi(x_t),$$

where $\ell_{t,i} \in \mathbb{R}^{|\mathcal{V}|}$ is the token logit vector at position i and H_t is the corresponding internal hidden representations. Each position has a state represented by

$$s_{t,i} \in \{\texttt{prompt}, \texttt{gen}, \texttt{locked}, \texttt{eot}\}.$$

`Prompt` positions are immutable. `Locked` positions are generated tokens that have already been accepted and will not be revised. Active `gen` positions are revisable generation positions on which

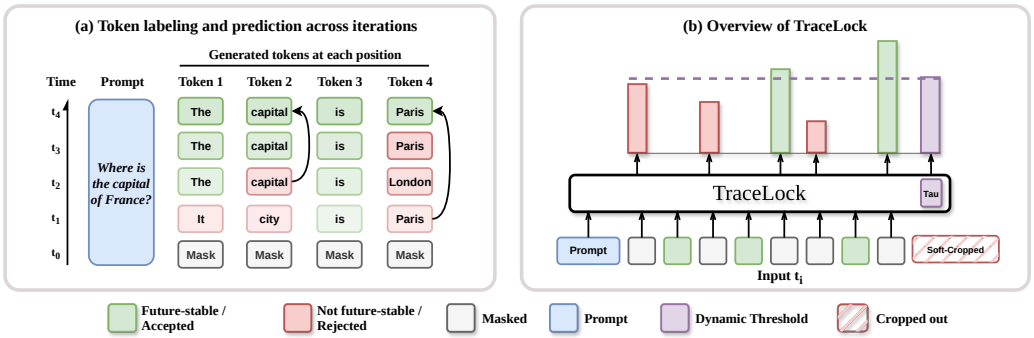


Figure 2: Overview of TRACELOCK. (a) Completed diffusion traces provide dense token-level supervision: an intermediate token is labeled future-stable if it matches the token at the same position in the final completed trace. At deployment, TRACELOCK predicts the same future-stability event to decide whether a proposed token should be accepted. Here, future stability refers to agreement with the final trace token rather than task-level answer correctness. (b) TRACELOCK scores active tokens from the current trace state and compares them with a sequence-conditioned dynamic threshold. Soft-cropping restricts commitment decisions to the active local window, while prompt tokens and previously accepted tokens remain as context.

the controller acts. The `eot` state marks generated end-of-text or padding positions that terminate the decoded answer and are removed by the tokenizer when forming the final response. The central decision is therefore a transition

$$\texttt{gen} \rightarrow \texttt{locked}.$$

Given the current candidate token

$$\hat{x}_{t,i} = \arg\max_{v \in \mathcal{V}} \ell_{t,i,v},$$

the controller decides whether to commit that candidate:

$$u_{t,i} \in \{0, 1\}, \qquad i \in \mathcal{G}_t,$$

where $\mathcal{G}_t = \{i : s_{t,i} = \texttt{gen}\}$. If $u_{t,i} = 1$, the position is filled with $\hat{x}_{t,i}$ and becomes `locked`; otherwise it remains masked for later refinement.

This formulation separates token proposal from token commitment. The D-LLM proposes candidate tokens; TRACELOCK only controls when candidates should stop being revised. The space of possible commitment traces is combinatorial: committing N positions across multiple non-empty rounds yields exponentially many possibilities. Exhaustively searching this trace space is infeasible, so our goal is to amortize the search by learning a policy that favors commitments already consistent with the eventual completed trace. TRACELOCK does not enlarge the expressive capacity of the base model; it biases decoding toward more favorable trajectories within the base model's reachable trace space.

3.2 LEARNING A COMMITMENT POLICY FROM FUTURE STABILITY

Self-supervised future-stability labels. For each completed trace, we observe the final sequence $x^\star = x_T$ and the candidate token $\hat{x}_{t,i}$ proposed by the D-LLM at intermediate steps. We define a dense future label

$$y_{t,i} = \mathbb{I}\left[\hat{x}_{t,i} = x_i^\star\right].$$

This label asks whether the current candidate has already reached its final trace value. It is not a gold-answer label or a reward-model label; it is a self-supervised trace signal derived from the model's own completed rollout. The target is weak because a token can temporarily match the final value and later change, but it is dense, cheap, and aligned with the deployment-time question: which active positions are stable enough to lock now?

Trace collection and filtering. To generate training traces, we run the frozen base model, record candidate tokens and intermediate hidden representations at selected steps, and backfill labels by comparing each recorded step against the final completed sequence. Trace quality matters: low-quality traces would produce a poor controller since the controller is trained to reproduce the stability patterns in these traces. We therefore generate the main supervised trace pool with relatively fine block schedules, which preserves quality at the cost of slow inference. This lets TRACELOCK learn from higher-quality traces and later deploy under larger local windows or longer generation regimes.

Furthermore, we also filter traces before training. For open-ended text, we remove generations that are too short or exhibit obvious degeneration, including malformed, grammatically invalid, or highly repetitive text, using lightweight rule-based grammar checks when applicable (Morris). For coding data, additional syntax and executable checks are used when available.

Feature representation. The controller is driven by hidden representations from the frozen D-LLM. For each position i at step t, let $\phi_{t,i}$ denote its trace-state feature which consists of an embedding of the last three hidden-states and differences between them, together with positional encodings. The intuition is that token stability is reflected not only in what the final hidden state is, but also in how the representation changes across late layers. Appendix Table 10 ablates these hidden-state components, and Table 12 compares hidden-state inputs with confidence-only token filtering.

Contextual token scoring. TRACELOCK uses a lightweight Transformer encoder (Vaswani et al., 2017) over the trace-state feature sequence. Given $\phi_{t,1:L}$, it returns one raw stability logit per position,

$$a_{t,i} = f_\theta(\phi_{t,1:L})_i.$$

The controller is not parameterized as an MLP over a fixed-size block of confidence scores; it is a shared sequence policy over token trace states. Consequently, the number of active positions considered at deployment may vary with the local window or generation length, while the learned parameters remain unchanged. Contextual encoding lets each token decision depend on prompt tokens, already locked positions, and still-active generation positions in the current partial sequence.

Dynamic threshold. A trace-independent threshold on token scores can be rigid: it may be too conservative for easy samples and too aggressive for difficult ones. TRACELOCK instead predicts a step-level threshold from the current trace state. We prepend a learned threshold token to the same feature sequence used for token scoring. After contextual encoding, a small threshold head applied to this token outputs a scalar threshold logit

$$\tau_t = g_\theta(\phi_{t,1:L}).$$

Here g_θ denotes the threshold branch of the TRACELOCK controller, not a separate language model. The decision logit for token i is the margin

$$m_{t,i} = a_{t,i} - \tau_t.$$

Training this margin creates a direct competition between positive and negative examples: stable tokens push the threshold below their scores, while unstable tokens push it above. Thus the learned threshold acts as an adaptive caution value conditioned on the current partial sequence and the hidden states produced at this decoding step. Appendix Figure 6 visualizes the learned threshold trajectories.

Training objective. The loss trains TRACELOCK to predict which tokens can be committed:

$$\mathcal{L}_{\text{dyn}} = \frac{1}{|\Omega|} \sum_{(t,i)\in\Omega} W(m_{t,i}, y_{t,i}) \,\mathrm{BCE}(m_{t,i}, y_{t,i}) + \lambda_\tau \tau_t^2,$$

$$W(m, y) = \begin{cases} 1, & \text{if } (m > 0) = y \\ k, & \text{if } (m > 0) \neq y. \end{cases}$$

where Ω runs over active generation positions excluding prompt and locked tokens,

$$\Omega = \{(t, i) : s_{t,i} = \texttt{gen}\}.$$

The first term trains future-stability prediction, while the second regularizes the learned threshold logit. The weighting factor $W(m_{t,i}, y_{t,i})$ is used to strongly penalize incorrect stability decisions, since incorrect commitments strongly degrade the final prediction. We set $k = 4$ in our experiments.

During training, we simulate varying window sizes with random prefix cropping. A visible generation prefix is sampled as

$$K = \mathrm{round}(rN), \qquad r \sim \mathrm{Uniform}(0.25, 1),$$

and masks attention beyond $P + K$. This simulates the partial contexts seen by local-window deployment and exposes the same controller to variable-length trace states during training.

3.3 Deploying the Learned Policy with TraceLock

At deployment, TraceLock uses a local soft-crop window rather than committing over the entire generation region at every step. Let $p_t = \min \mathcal{G}_t$ be the first still-active generation position and let w be the window width. The controller acts on window

$$W_t = [p_t, \min(p_t + w - 1, L)]$$

where w is the desired window length. Operationally, this is a two-pointer policy: a slow pointer tracks the first unresolved position, while the fast right pointer defines the active decision window. This avoids strict divisibility constraints from fixed block decoding and keeps the candidate set from shrinking too sharply near block boundaries.

Within the active window, TraceLock converts the learned margin into a commit probability and commits positions satisfying $\mathrm{sigmoid}(m_{t,i}) > \tilde{\tau}$, where $\tilde{\tau}$ is a fixed global operating threshold rather than a per-setting calibration parameter. We set $\tilde{\tau} = 0.95$ for all main experiments, making the controller conservative against incorrect early commitments. If no position passes the threshold, the policy commits the highest-scoring active position as a fallback. This prevents the decoding loop from stalling while still allowing high-stability tokens to be accepted early. Because the controller scores the currently exposed trace state rather than a fixed block interface, the same checkpoint can be deployed under different window widths, generation lengths, and step budgets. Table 2 and Figure 4 evaluate this deployment flexibility.

4 Experiments

We evaluate TraceLock as an inference-time commitment controller for frozen diffusion language models. The experiments test three claims: whether a learned trace-state policy improves the quality–step tradeoff over heuristic and learned baselines, whether the same controller can be deployed across local-window and generation-length changes without retraining, and whether its decisions differ from confidence filtering. We evaluate these questions on mathematical reasoning, question answering, and code generation.

4.1 Experimental Setup

Backbones. All decoding policies are evaluated on frozen D-LLM backbones. We use LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) as the frozen backbones.

Tasks. Training traces are collected from the frozen backbone and then labeled by future stability, as described in Section 3.2. The trace pool covers GSM8K for math (Cobbe et al., 2021), Natural Questions and Alpaca-style instruction data for QA (Kwiatkowski et al., 2019; Taori et al., 2023), and KodCode for coding (Xu et al., 2025). Evaluation uses GSM8K final-answer accuracy for math, Alpaca-style QA prompts for answer ranking, and HumanEval execution tests for code generation (Chen et al., 2021).

Baselines. We compare TraceLock with policies that use the same frozen backbone: random transfer, confidence transfer, Fast-dLLM-style thresholded decoding (Wu et al., 2025b;a), and Learn2PD (L2P) (Bao et al., 2026). Each Learn2PD backbone/block-size setting uses 2,640 fresh traces and a newly trained model, following the original implementation. The default TraceLock deployment uses the local soft-crop window and sequence-conditioned dynamic threshold from Section 3.3. Appendix Table 13 reports optional self-training and RL adaptation variants.

Set	Rand.	Conf.	Fast	L2P	TL
Math (ACC ↑)					
128/32	67.1	74.4	74.8	74.8	**75.6**
256/64	69.1	79.8	79.4	78.3	**80.0**
QA (Avg. rank ↓)					
128/32	3.42	2.89	3.06	3.02	**2.60**
256/64	3.51	2.74	2.87	3.21	**2.68**
Coding (Pass@1 ↑)					
128/32	14.0	31.1	29.9	28.7	**34.2**
256/64	15.2	36.0	35.4	32.3	**37.8**

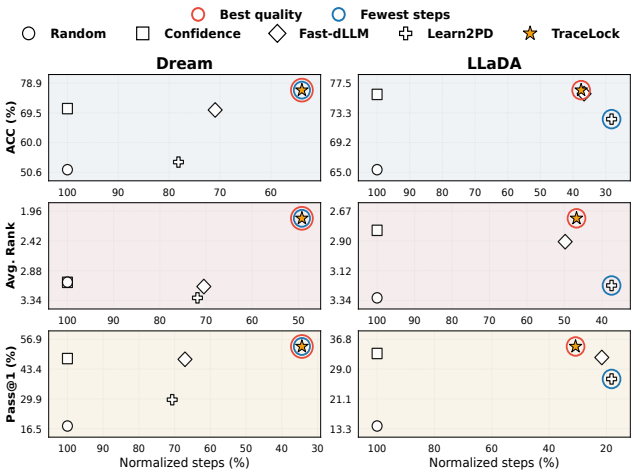


Table 1: Compact main-result summary and average quality–step tradeoff. Left: scores with LLaDA using generation length / window-or-block size $128/32$ and $256/64$ over the three tasks. The full table is available in Appendix Table 3. Right: plots of average scores versus normalized number of steps taken; fewer steps indicate faster sampling. Higher scores are better for Math and Coding, while lower scores are better for QA average rank.

Metrics. For math, a response is correct if its extracted final answer matches the GSM8K target. For coding, we report HumanEval Pass@1; syntax errors, runtime errors, and timeouts are counted as failures. For QA, we anonymize the candidate answers for the same prompt, rank them in a single batch with Llama-3.1-8B-Instruct (Grattafiori et al., 2024) under a shared rubric, and report average rank, where lower is better. All QA average-rank results in the paper use this within-prompt batch-ranking protocol, but ranks are comparable only within the candidate set used by each table. Efficiency is measured by average executed decoding steps.

4.2 Main Results

Table 1 summarizes the main quality–step tradeoff. The left panel shows a compact LLaDA slice at generation length / block size $128/32$ and $256/64$, while the right panel reports six model–task averages: for each backbone and task, primary metrics are averaged over the tested length/window-or-block size settings and steps are first normalized by the maximum scheduled steps for that generation length before averaging. Appendix Table 3 reports every tested length/window setting for both backbones, and Figure 4 gives denser generation-length and local-window sweeps on HumanEval using the same trained controller. Together, these results substantiate the main-table conclusions across multiple deployment settings and show that the learned policy remains effective beyond the two settings highlighted in the compact table.

Overall, TraceLock improves the quality–step frontier, but its operating point differs across backbones. On Dream, it often improves both task score and decoding steps relative to the strongest baselines. On LLaDA, it typically prioritizes final quality, achieving the best or near-best scores while using more decoding steps than the most aggressive baselines. These results support the view that token commitment can be learned as a reusable policy over diffusion trace states.

Other Variations Beyond the default frozen-controller setting (TraceLock), we also explored two optional extensions: (1) target distribution adaptation (TraceLock-ST) and (2) reinforcement-learning-based refinement (TraceLock-RL). These extensions are not required for the main results, but suggest that the learned commitment policy can be further improved with additional adaptation (Appendix H and Table 13). TraceLock-ST mainly reduces the number of denoising steps, while TraceLock-RL can further improve final task performance in selected settings.

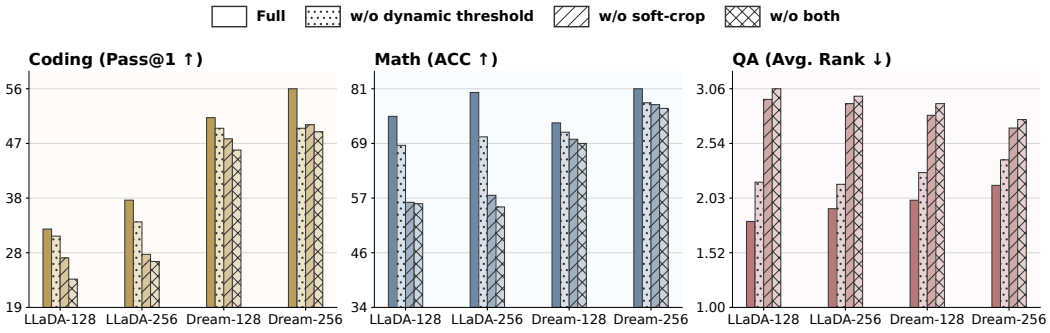


Figure 3: Deployment-mechanism ablations by domain. Each panel shows the full TRACELOCK deployment, removal of soft-crop, replacement of the dynamic threshold with a static threshold, and removal of both mechanisms. Full numbers are in Appendix Table 11.

4.3 ABLATION STUDY

TRACELOCK's default deployment uses two implementation choices: soft-crop defines the active decision window, while the learned sequence-conditioned threshold provides a trace-conditioned acceptance boundary. Figure 3 ablates these mechanisms across coding, math, and QA. The full controller is consistently better than removing either choice, and removing both is the weakest setting overall. This indicates that the learned future-stability policy benefits from both a local decision scope and a trace-conditioned acceptance boundary.

4.4 COMPARISON TO BLOCKWISE AND CONFIDENCE-BASED COMMITMENT

Goal of comparison. The closest alternatives to TRACELOCK are learned blockwise filters such as Learn2PD and training-free confidence-based transfer rules. Both decide which tokens should be accepted during diffusion decoding, but they induce different inductive biases. Learn2PD learns a filter tied to a fixed block interface, while confidence-based methods use scalar token confidence as the main acceptance signal. We therefore use this section to test two diagnostic questions: whether TRACELOCK remains stable when the deployment window changes, and whether its trajectories differ from confidence filtering on the same prompts.

Reusability beyond a fixed block interface. A practical D-LLM decoder should remain usable when the generation length or local-window size changes, since both are deployment choices determined by the task length and latency budget. Under Learn2PD protocol, each block size requires new confidence traces and a separately trained MLP because its input-output interface is tied to that block shape. In contrast, TRACELOCK scores variable-length trace states and can therefore be deployed under different local windows without changing the policy architecture. Table 2 compares Learn2PD and TRACELOCK when the test-time block or local window is enlarged. To test transfer to a doubled window without collecting new traces or retraining, we apply Learn2PD's fixed-window filter independently to two original-size subwindows. Across GSM8K, Alpaca-style QA, and HumanEval, Learn2PD often degrades whereas TRACELOCK remains more stable.

Figure 4 further sweeps TRACELOCK over generation lengths and window sizes without retraining. Performance remains broadly stable across both backbones. These deployments use the same global acceptance cutoff $\tilde{\tau} = 0.95$ on the learned margin, with no setting-specific cutoff search. Appendix Table 7 separately shows that changing this cutoff trades task score for executed steps.

Additional comparisons to confidence-based policies. Confidence is still a natural reference point for any token-commitment policy, since it is a standard training-free signal available from the base diffusion model. We therefore include several appendix diagnostics that compare TRACELOCK against confidence-based behavior beyond final task scores. Table 14 measures trajectory-level divergence from confidence filtering on matched QA prompts; Figure 7 compares TRACELOCK scores with token confidence over decoding time; Figures 8 and 9 visualize the two policies on

Table 2: Cross-configuration reuse from the original block/window to a larger one. GSM8K reports accuracy (%), QA reports average rank under four-candidate within-prompt ranking, and HumanEval reports Pass@1 (%). Parentheses show the change from original to generalized; for QA, lower rank is better, so positive rank changes indicate degradation.

Method	Gen len	Block change	LLaDA		Dream	
			Original	Generalized	Original	Generalized
Learn2PD	128	32→64	74.8	64.4 (-10.4 pp)	69.1	66.8 (-2.3 pp)
TraceLock	128	32→64	75.60	74.0 **(-1.6 pp)**	73.5	71.5 **(-2.0 pp)**
Learn2PD	256	64→128	78.3	51.1 (-27.2 pp)	64.6	45.2 (-19.4 pp)
TraceLock	256	64→128	80.0	76.5 **(-3.5 pp)**	80.8	80.7 **(-0.1 pp)**
Learn2PD	128	32→64	2.28	2.77 (+0.49 rank)	2.65	3.14 (+0.49 rank)
TRACELOCK	128	32→64	2.07	2.18 **(+0.12 rank)**	1.81	2.05 **(+0.24 rank)**
Learn2PD	256	64→128	2.33	2.71 (+0.38 rank)	2.62	3.15 (+0.53 rank)
TRACELOCK	256	64→128	2.03	2.23 **(+0.20 rank)**	1.82	2.14 **(+0.32 rank)**
Learn2PD	128	32→64	28.7	25.0 (-3.7 pp)	48.8	45.7 (-3.1 pp)
TRACELOCK	128	32→64	34.15	32.32 **(-1.83 pp)**	51.2	51.2 **(+0.0 pp)**
Learn2PD	256	64→128	32.3	23.2 (-9.1 pp)	31.7	28.0 (-3.7 pp)
TRACELOCK	256	64→128	37.80	35.4 **(-2.4 pp)**	56.1	56.1 **(+0.0 pp)**

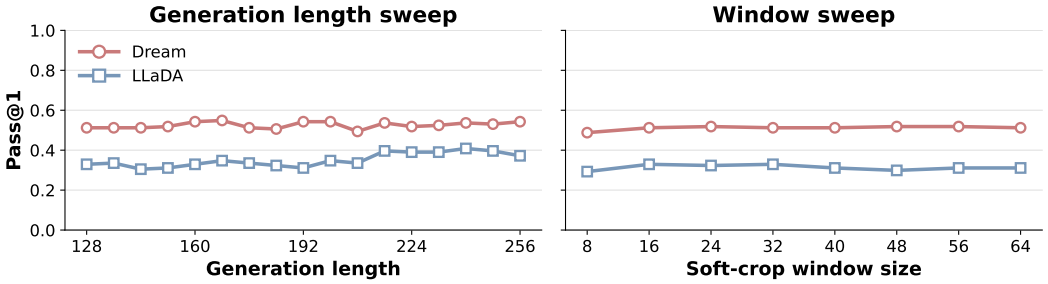


Figure 4: Generation-length and pointer-window sweeps on HumanEval. Each point is a distinct decoding configuration of the same trained TRACELOCK controller; performance remains stable across the tested range.

matched qualitative traces; and Table 12 replaces confidence inputs in a Learn2PD-style blockwise filter with hidden-state features. Together, these diagnostics show that TRACELOCK remains related to confidence but is not simply a different threshold on the same confidence ordering.

5 CONCLUSION

Findings. We presented TRACELOCK, a learned token-commitment policy for frozen masked diffusion language models. Future-stability labels from completed decoding traces provide dense supervision over variable-length trace states. Across mathematical reasoning, question answering, and code generation, TRACELOCK improves the quality–step tradeoff over heuristic and learned decoders. Controlled single-token interventions further show that selected commitments are usually consistent with the subsequent reference continuation and rarely harm task correctness. The same controller remains effective across local-window widths and generation lengths without retraining, supporting token commitment as a reusable learned policy over diffusion trace states.

Limitations and scope. Future-stability labels measure agreement with the model's completed rollouts, not external task correctness, and can inherit biases from those traces. Our evidence concerns fixed frozen-generator families and related decoding interfaces; transfer to unrelated generators, sampling procedures, or task distributions remains untested. Judge-based QA evaluation complements automatically checkable metrics such as GSM8K accuracy and HumanEval execution.

AI USE STATEMENT

Large language models assisted with literature organization, manuscript drafting and language editing, and code inspection and debugging. The authors designed the method and experiments, verified the reported results and citations, and reviewed the final manuscript. LLM-based QA judging and reward-model scoring used within the experiments are described separately in the relevant sections.

ETHICS STATEMENT

This work studies decoding policies for frozen language models using existing research datasets and model-generated outputs; it does not involve new data collection involving human subjects. The controller may inherit errors and biases from the backbone models and training traces, while LLM-based QA judgments may introduce evaluator bias. Faster decoding may also reduce the cost of generating harmful content. We do not claim that TRACELOCK improves output safety.

REPRODUCIBILITY STATEMENT

The supervision target, controller architecture, training objective, and decoding procedure are described in Section 3.2 and Section 3.3. Section 4 specifies the backbones, datasets, baselines, metrics, and evaluated settings. The appendix provides further details on trace collection, controller training, compute resources, ablations, and diagnostic analyses.

REFERENCES

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. *ICLR*, 2025.

Parikshit Bansal and Sujay Sanghavi. Enabling approximate joint sampling in diffusion lms. *arXiv:2509.22738*, 2025.

Wenrui Bao, Zhiben Chen, Dan Xu, and Yuzhang Shang. Learning to parallel: Accelerating diffusion large language models via learnable parallel decoding. *International Conference on Learning Representations*, 2026.

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, 2021.

Xinhua Chen, Sitao Huang, Cong Guo, Chiyue Wei, Yintao He, Jianyi Zhang, Hai Li, Yiran Chen, et al. Dpad: Efficient diffusion language models with suffix dropout. *arXiv preprint arXiv:2508.14148*, 2025a.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. dparallel: Learnable parallel decoding for dllms. *arXiv:2509.26488*, 2025b.

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.

Yihong Dong, Zhaoyu Ma, Xue Jiang, Zhiyuan Fan, Jiaru Qian, Yongmin Li, Jianha Xiao, Zhi Jin, Rongyu Cao, Binhua Li, et al. Saber: An efficient sampling with adaptive acceleration and backtracking enhanced remasking for diffusion language model. *arXiv:2510.18165*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, et al. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *arXiv:2507.18578*, 2025.

Yuezhou Hu, Harman Singh, Monishwaran Maheswaran, Haocheng Xi, Coleman Hooper, Jintao Zhang, Aditya Tomar, Michael W. Mahoney, Sewon Min, Mehrdad Farajtabar, Kurt Keutzer, Amir Gholami, and Chenfeng Xu. Residual context diffusion language models. *arXiv:2601.22954*, 2026.

Zemin Huang, Yuhang Wang, Zhiyang Chen, and Guo-Jun Qi. Don't settle too early: Self-reflective remasking for diffusion language models. *arXiv:2509.23653*, 2025.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, and Xu Yang. d^2 cache: Accelerating diffusion-based llms via dual adaptive caching. *arXiv:2509.23094*, 2025.

Bumjun Kim, Dongjae Jeon, Moongyu Jeon, and Albert No. Dependency-aware parallel decoding via attention for diffusion llms. *arXiv:2603.12996*, 2026.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *ICML*, 2025.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 2019.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language models know the answer before decoding. *arXiv:2508.19982*, 2025.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*, 2025.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The flan collection: Designing data and methods for effective instruction tuning, 2023. URL https://arxiv.org/abs/2301.13688.

Jack Morris. language_tool_python. https://pypi.org/project/language-tool-python/. Accessed: 2026-05-06.

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. *NeurIPS*, 2025.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *ICLR*, 2025.

Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *NeurIPS*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv:2402.03300*, 2024.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Wen Wang, Bozhen Fang, Chenchen Jing, Yongliang Shen, Yangyi Shen, Qiuyu Wang, Hao Ouyang, Hao Chen, and Chunhua Shen. Time is a feature: Exploiting temporal dynamics in diffusion language models. *arXiv:2508.09138*, 2025a.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv:2508.09192*, 2025b.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022. URL https://arxiv.org/abs/2109.01652.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. *arXiv:2509.26328*, 2025a.

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. *arXiv:2505.22618*, 2025b.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv:2503.02951*, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv:2508.15487*, 2025.

Yifan Yu, Yuqing Jian, Junxiong Wang, Zhongzhu Zhou, Donglin Zhuang, Xinyu Fang, Sri Yanamandra, Xiaoxia Wu, Qingyang Wu, Shuaiwen Leon Song, Tri Dao, Ben Athiwaratkun, James Zou, Fan Lai, and Chenfeng Xu. Introspective diffusion language models. *arXiv:2604.11035*, 2026.

A Detailed Related Work

A.1 Implemented baselines.

Random and confidence transfer. We include the two native remasking strategies used by the LLaDA and Dream sampling code: random transfer and confidence transfer (Nie et al., 2025; Ye et al., 2025). Generation follows a semi-autoregressive (Semi-AR) schedule: the output region is divided into local blocks, and decoding proceeds block by block from left to right. At each denoising step, the frozen D-LLM proposes tokens for the currently masked positions in the active block. For stability, we transfer one position per step for both random and confidence transfer. Random transfer selects this position uniformly at random from the active block. Confidence transfer selects the most confident candidate in the active block. For LLaDA, confidence is the softmax probability of the proposed token; for Dream, we follow the native implementation and use its entropy-based confidence score.

Fast-dLLM. Fast-dLLM is a stronger confidence-based decoding baseline that uses thresholded token transfer (Wu et al., 2025b;a). At each step, candidate tokens whose confidence exceeds a fixed threshold are transferred immediately. If no candidate reaches the threshold, the policy falls back to transferring the highest-confidence token, preventing the decoding process from stalling. Following the public Fast-dLLM implementation, we set the threshold to 0.9 in all experiments.

Learn2PD. We also include Learn2PD, a learned parallel-decoding filter for frozen D-LLMs (Bao et al., 2026). Following the original implementation, we train a separate confidence filter for each backbone and block size in $\{\text{LLaDA}, \text{Dream}\} \times \{32, 64, 128\}$. To match the original Learn2PD setup, we use the FLAN instruction-tuning data for filter training (Wei et al., 2022; Longpre et al., 2023). For each backbone–block-size setting, we sample 100 prompts per task, newly collect 2,640 confidence traces with the corresponding frozen backbone, and train a fresh two-layer MLP filter. Thus, the matched-setting Learn2PD results use a model trained for that block size; the cross-block transfer test in Table 2 instead reuses the original block-size model at the larger block size. At inference time, we use the thresholds from the original codebase: 0.96 for LLaDA and 0.9 for Dream.

A.2 Other related works

Prophet asks whether a D-LLM already "knows" the answer before all scheduled denoising steps are complete and uses this observation for early commit or early stopping (Li et al., 2025). It is highly relevant because it frames decoding as a commitment problem, but its main decision is sequence-level stopping rather than token-level contextual locking. We therefore treat it as a potential future baseline for early-exit behavior, not as a direct replacement for a token controller.

APD uses a small auxiliary autoregressive model to guide adaptive parallel decoding (Israel et al., 2025). It is related because it learns an adaptive decoding behavior at inference time, but it changes the inference stack by adding a second generator. This makes it less direct for our frozen-D-LLM-only setting, where the goal is to learn from the hidden states and traces of the target D-LLM itself.

DAPD is a training-free dependency-aware method that uses attention to infer which masked tokens can be decoded in parallel (Kim et al., 2026). It is complementary to TraceLock: DAPD uses an explicit dependency graph, while TraceLock learns a stability score from trace states. We do not include it in the current experiments because our present baseline suite focuses on confidence and learned-controller comparisons; dependency-aware scheduling is a natural future addition.

dParallel adapts the generator through distillation so that the model becomes more suitable for parallel decoding (Chen et al., 2025b). This is an important generator-adaptation baseline, but it answers a different question from TraceLock. Our method asks what can be gained by learning an inference-time policy on top of a fixed model, whereas dParallel changes the model being decoded.

RemeDi adds remasking ability to the model itself through remask-aware supervised finetuning and reinforcement learning (Huang et al., 2025). It is closely related at the level of motivation because both methods care about premature commitment. The key difference is that RemeDi trains a D-LLM to revise tokens, while TraceLock leaves the generator frozen and controls when proposed tokens are locked.

D2F and **Block Diffusion** change the generation geometry by combining autoregressive and diffusion-style behavior (Wang et al., 2025b; Arriola et al., 2025). These methods can improve the speed–quality tradeoff, but they operate through a different backbone or decoding formulation. We therefore view them as complementary generator-design baselines rather than direct plug-in controller baselines.

RCD argues that confidence-based remasking wastes computation by discarding unresolved token representations, then recycles those residual representations into later denoising steps (Hu et al., 2026). This is conceptually aligned with our claim that trace state matters beyond local confidence. However, RCD modifies the model architecture or training pipeline, while TraceLock only learns an auxiliary commitment policy.

I-DLM introduces an introspective diffusion-language-model paradigm with verification-like acceptance during decoding (Yu et al., 2026). It is relevant because it explicitly studies acceptance and self-consistency, but it is not a plug-in policy for an existing frozen D-LLM. Comparing against it would require a different model family and serving stack.

Finally, systems and execution-level methods such as **Fast-dLLM v2**, **d^2 Cache**, and **DPad** reduce the cost of each denoising step through caching, suffix-token pruning, or attention-computation optimizations(Wu et al., 2025a; Jiang et al., 2025; Chen et al., 2025a). These methods are orthogonal to TraceLock: a learned commitment policy can in principle be combined with lower-cost model execution. For this reason, our current experiments focus on decoding-policy comparisons rather than systems-level throughput engineering.

B Full Main Results

Table 3 reports the full per-setting numbers summarized in Table 1.

Additional QA ranking and critic metrics. Table 4 reports two complementary QA statistics for the same policy groups and decoding settings as the QA block in Table 3. The top-two rate is the percentage of evaluated samples whose candidate is placed in the top-two group for the prompt. The critic rate uses the Alpaca reference output as a non-exhaustive reference answer and asks an

Table 3: Combined main results across math, QA, and coding. Within each domain block, each policy is reported with a primary task metric and the average number of executed decoding steps. Lower is better only for QA average rank. Appendix Table 4 gives additional QA top-two and critic metrics.

Model	Gen len	Win./Block	Random		Confidence		Fast-dLLM		L2P		TraceLock	
Math (ACC ↑)			ACC (%) ↑	Avg. step ↓	ACC (%) ↑	Avg. step ↓	ACC (%) ↑	Avg. step ↓	ACC (%) ↑	Avg. step ↓	ACC (%) ↑	Avg. step ↓
LLaDA	128	32	67.1	128.0	74.4	128.0	74.8	54.3	74.8	**44.7**	**75.60**	50.09
	128	64	62.5	128.0	73.5	128.0	73.5	53.4	70.5	**40.7**	**74.0**	50.58
	256	64	69.1	256.0	79.8	256.0	79.4	79.4	78.3	**57.4**	**80.00**	85.78
	256	128	62.7	256.0	76.0	256.0	76.4	80.4	66.2	**60.9**	**76.5**	97.4
Dream	128	32	59.5	128.0	**73.8**	128.0	73.2	**105.5**	69.1	117.4	73.5	107.5
	128	64	51.2	128.0	70.0	128.0	69.5	87.6	57.3	111.2	**71.5**	**58.4**
	256	64	51.9	256.0	79.9	256.0	79.9	197.4	64.6	203.5	**80.8**	**106.1**
	256	128	43.2	256.0	59.4	256.0	58.8	143.0	24.4	139.8	**80.7**	**114.1**

Model	Gen len	Win./Block	Random		Confidence		Fast-dLLM		L2P		TraceLock	
QA (average rank ↓)			Avg. rank ↓	Avg. step ↓	Avg. rank ↓	Avg. step ↓	Avg. rank ↓	Avg. step ↓	Avg. rank ↓	Avg. step ↓	Avg. rank ↓	Avg. step ↓
LLaDA	128	32	3.42	128.0	2.89	128.0	3.06	73.0	3.02	**64.6**	**2.60**	66.7
	128	64	3.40	128.0	2.83	128.0	**2.81**	69.6	3.10	**51.7**	2.87	61.0
	256	64	3.51	256.0	2.74	256.0	2.87	117.3	3.21	**90.5**	**2.68**	116.3
	256	128	2.96	256.0	2.81	256.0	2.87	107.5	3.59	**60.1**	**2.76**	107.3
Dream	128	32	3.38	128.0	2.96	128.0	3.09	106.2	3.13	109.3	**2.31**	**66.0**
	128	64	3.12	128.0	3.04	128.0	3.19	83.5	3.23	89.2	**2.02**	**69.6**
	256	64	3.09	256.0	3.06	256.0	3.01	201.8	3.28	205.7	**2.20**	**111.4**
	256	128	2.62	256.0	3.16	256.0	3.19	140.6	3.54	133.2	**1.73**	**121.9**

Model	Gen len	Win./Block	Random		Confidence		Fast-dLLM		L2P		TraceLock	
Coding (Pass@1 ↑)			Pass@1 (%) ↑	Avg. step ↓	Pass@1 (%) ↑	Avg. step ↓	Pass@1 (%) ↑	Avg. step ↓	Pass@1 (%) ↑	Avg. step ↓	Pass@1 (%) ↑	Avg. step ↓
LLaDA	128	32	14.0	128.0	31.1	128.0	29.9	33.5	28.7	**29.2**	**34.15**	40.24
	128	64	11.6	128.0	31.1	128.0	28.7	33.2	24.4	**28.0**	**32.32**	40.62
	256	64	15.2	256.0	36.0	256.0	35.4	44.9	32.3	**35.2**	**37.80**	80.33
	256	128	15.2	256.0	34.1	256.0	34.1	43.2	20.1	**37.1**	**35.4**	73.22
Dream	128	32	22.6	128.0	47.0	128.0	45.7	100.6	48.8	106.6	**51.2**	**47.9**
	128	64	19.5	128.0	48.8	128.0	47.6	77.9	32.9	91.0	**51.2**	**47.7**
	256	64	13.4	256.0	52.4	256.0	53.0	195.4	31.7	197.5	**56.1**	**79.9**
	256	128	15.2	256.0	44.5	256.0	45.1	134.7	4.9	131.0	**56.1**	**80.9**

Table 4: Additional QA metrics on the same settings as the main QA table. Each policy is reported with top-two rate and LLM-critic rate, both in percent. Higher is better for both metrics.

Model	Gen len	Win./Block	Random		Confidence		Fast-dLLM		L2P		TraceLock	
QA			Top-2 ↑	Critic ↑	Top-2 ↑	Critic ↑	Top-2 ↑	Critic ↑	Top-2 ↑	Critic ↑	Top-2 ↑	Critic ↑
LLaDA	128	32	36.0	87.0	47.0	90.4	41.0	88.9	42.5	81.2	**49.0**	**91.2**
	128	64	34.5	**85.8**	**52.6**	84.8	51.0	83.8	43.0	84.3	52.0	83.9
	256	64	31.5	84.0	57.5	81.7	51.5	85.9	35.5	85.1	**58.0**	**87.4**
	256	128	44.0	82.4	49.5	80.5	46.0	82.8	25.1	79.1	**50.8**	**85.7**
Dream	128	32	37.4	82.6	40.5	**84.0**	33.7	80.9	33.2	83.1	**67.3**	83.5
	128	64	45.4	82.0	37.2	77.8	31.4	84.7	27.4	83.7	**63.4**	**85.3**
	256	64	48.4	83.1	37.5	78.6	39.2	75.0	28.1	**84.3**	**69.2**	83.1
	256	128	57.1	69.9	38.2	75.4	38.9	64.1	25.4	72.7	**63.0**	**79.3**

LLM critic to judge whether the generated answer should be considered correct against that reference. Examples marked not judgeable are skipped in the critic denominator.

These QA critic numbers are only auxiliary. Alpaca-style prompts include many open-ended requests, and a single reference output is often only one acceptable answer. For stronger correctness claims, automatically checkable settings such as GSM8K math accuracy and HumanEval execution remain more reliable.

C Additional Validation and Sensitivity Analyses

C.1 Counterfactual Validation of Token Commitment

The future-stability target is observational: a proposal agrees with the terminal token of a recorded trajectory, but committing that proposal earlier may change the subsequent trajectory. We therefore evaluate a controlled intervention that isolates the effect of one additional TraceLock-selected commitment.

For each eligible GSM8K prompt, we identify a state in which TraceLock would commit a token that the native confidence-based decoder would leave masked. We select one such event per prompt using only information available before the intervention. From the same state, the reference branch R applies the native decoder's commitments, while the intervention branch L applies those same commitments plus the selected token. Both branches then continue with the native decoder, without further TraceLock interventions. The additional commitment is thus the only exogenous difference between the branches.

We measure three quantities. *Counterfactual lock agreement* is the fraction of selected proposals that match the final reference token at the selected position. *Other-positive-label retention* is the fraction of other current proposals matching the final R sequence that also match the final L sequence. *Task-harm rate* is the fraction of reference-correct events for which the intervention changes the final GSM8K answer to an incorrect one.

We evaluate one event from each of 500 eligible prompts per backbone. The frozen backbones are LLaDA-8B-Instruct and Dream-v0-Instruct-7B, with generation length / scheduled steps / window size of $256/256/64$ and $128/128/32$, respectively. Table 5 reports the results. Lock-agreement and task-harm intervals are two-sided 95% Wilson intervals; retention intervals use a 10,000-replicate bootstrap clustered by prompt.

Table 5: Single-lock interventions on GSM8K. Values are percentages with 95% confidence intervals. Counts below each estimate give the relevant denominator: selected events, other reference-positive positions, or reference-correct events. Higher agreement and retention are better; lower task harm is better.

Backbone	Lock agreement	Other-positive retention	Task harm
LLaDA	89.00 [85.95, 91.45]	96.25 [94.99, 97.37]	0.545 [0.150, 1.965]
	445/500 events	19,265 positions; 500 events	2/367 reference-correct events
Dream	90.00 [87.06, 92.33]	95.72 [94.25, 97.03]	0.619 [0.170, 2.229]
	450/500 events	9,685 positions; 500 events	2/323 reference-correct events

Approximately nine out of ten selected proposals match the token reached by the reference continuation, and more than 95% of the other reference-positive proposals are preserved. Only two reference-correct events per backbone become incorrect after the intervention. These results provide evidence that the selected commitments are rarely harmful in this controlled setting. They do not equate terminal-token agreement with task correctness or establish the safety of repeated commitments: the experiment conditions on eligible events and intervenes on only one token before returning to native decoding.

C.2 Training-Objective Ablation

We ablate the two coefficients in the training objective: misclassification reweighting and threshold regularization. Setting $k = 1$ makes $W(m, y) = 1$ for all examples, removing the additional penalty on misclassified stability decisions. Setting $\lambda_\tau = 0$ removes the penalty on the learned threshold logit. These are training-objective changes, distinct from replacing the dynamic threshold with a static cutoff at inference time. Results are in Table 6.

Removing misclassification reweighting yields more aggressive commitment, but task scores drop by 10.00–22.56 percentage points. Removing threshold regularization has a smaller effect, reducing task

Table 6: Training-objective ablations on LLaDA. Each cell reports task score (%) / average executed steps. GSM8K uses final-answer accuracy and HumanEval uses Pass@1. The highest task score in each column is bolded.

	GSM8K		HumanEval	
Training configuration	$128/32$	$256/64$	$128/32$	$256/64$
w/o misclassification reweighting ($k = 1$)	57.80 / 28.95	70.00 / 56.13	20.12 / 28.18	15.24 / 55.64
w/o threshold regularization ($\lambda_\tau = 0$)	74.20 / 51.21	78.40 / 92.79	31.71 / 41.98	34.76 / 73.00
Full setup	**75.60** / 50.09	**80.00** / 85.78	**34.15** / 40.24	**37.80** / 80.33

scores by 1.40–3.04 points. The full setup achieves the highest task score in all four settings and uses fewer steps than the unregularized controller in three of them.

C.3 Sensitivity to the Global Acceptance Cutoff

The learned dynamic threshold τ_t and the global acceptance cutoff $\tilde{\tau}$ serve different roles. The controller predicts τ_t from the current trace state and forms the margin $m_{t,i} = a_{t,i} - \tau_t$. The deployment rule then accepts proposals satisfying $\mathrm{sigmoid}(m_{t,i}) > \tilde{\tau}$.

Table 7 reports the four TraceLock cutoff values $\tilde{\tau} \in \{0.50, 0.70, 0.90, 0.95\}$ on GSM8K and HumanEval for both backbones at generation length / window-or-block size $256/64$. The LLaDA entries use the updated controller. We compare against Learn2PD (L2P) at cutoffs 0.50 and 0.70, together with its official default cutoff for each backbone (0.96 for LLaDA and 0.90 for Dream).

Table 7: Global acceptance-cutoff sensitivity and comparison with L2P at generation length / window-or-block size $256/64$. Each task column reports score (%) / average executed steps. Scores are GSM8K final-answer accuracy and HumanEval Pass@1. "Default" marks the cutoff prescribed by the L2P implementation for each backbone. Bold indicates the highest task score within each backbone and task.

Backbone	Method	Cutoff	GSM8K	HumanEval
LLaDA	TraceLock	0.50	75.20 / 68.66	22.20 / 62.12
		0.70	77.20 / 74.67	27.00 / 73.48
		0.90	77.80 / 80.90	33.66 / 80.26
		0.95	**80.00** / 85.78	**37.80** / 80.33
	L2P	0.50	63.40 / 36.80	5.49 / 22.60
		0.70	67.60 / 44.13	8.54 / 26.61
		0.96 (default)	78.30 / 57.40	32.30 / 35.20
Dream	TraceLock	0.50	73.00 / 71.23	34.39 / 69.52
		0.70	77.80 / 76.94	39.63 / 71.96
		0.90	79.00 / 87.97	48.78 / 75.16
		0.95	**80.80** / 106.56	**56.09** / 79.94
	L2P	0.50	36.00 / 191.70	11.59 / 190.60
		0.70	50.80 / 195.40	17.07 / 192.47
		0.90 (default)	64.60 / 203.50	31.70 / 197.50

Across the four TraceLock backbone–task settings, both task score and executed steps increase monotonically with the cutoff. The shared operating point $\tilde{\tau} = 0.95$ gives the highest observed task score, while lower cutoffs trade quality for fewer model executions. The comparison supports a backbone-dependent quality–step tradeoff rather than uniform speed dominance.

Table 8: Controller-overhead profile on 100 fixed GSM8K samples at generation length 256 and window/block size 128. Control overhead is the control-module cost divided by end-to-end latency.

Backbone	Method	Latency (s/sample)	Throughput (samples/s)	Control cost (s/sample)	Overhead
Dream	TRACELOCK	13.391	0.0747	0.151	1.13%
Dream	L2P	12.483	0.0801	0.0163	0.131%
LLaDA	TRACELOCK	8.594	0.1164	0.163	1.90%
LLaDA	L2P	6.471	0.1545	0.078	1.2%

Table 9: Average wall-clock generation time in seconds. In each domain block, rows above the midrule are LLaDA results and rows below the midrule are Dream results. TRACELOCK* applies the correction factor defined above. Lower is better.

Math

Gen	Block	Confidence	Fast-dLLM	L2P	TRACELOCK	TRACELOCK*
128	32	9.10	4.14	**3.14**	3.85	3.59
128	64	8.89	4.07	**2.86**	3.78	3.52
256	64	24.07	8.01	**7.28**	8.45	8.13
256	128	23.31	8.10	**7.60**	9.49	9.14
128	32	6.41	6.16	6.02	6.69	**5.44**
128	64	6.40	4.41	5.70	3.64	**2.96**
256	64	19.45	14.74	15.44	9.76	**7.76**
256	128	18.96	10.70	10.63	10.49	**8.35**

QA

Gen	Block	Confidence	Fast-dLLM	L2P	TRACELOCK	TRACELOCK*
128	32	6.11	3.77	**3.08**	3.50	3.26
128	64	6.11	3.59	**2.47**	3.19	2.97
256	64	20.67	10.37	**7.34**	10.07	9.69
256	128	20.68	9.53	**4.85**	9.30	8.95
128	32	5.52	4.58	4.74	3.53	**2.87**
128	64	5.51	3.60	3.88	3.70	**3.01**
256	64	16.46	13.02	13.27	8.76	**6.97**
256	128	16.45	9.07	8.62	9.57	**7.61**

Coding

Gen	Block	Confidence	Fast-dLLM	L2P	TRACELOCK	TRACELOCK*
128	32	16.00	4.86	**4.37**	5.68	5.30
128	64	16.01	4.87	**4.14**	5.60	5.22
256	64	38.56	8.05	**6.27**	8.37	7.87
256	128	39.47	7.94	**6.62**	8.13	7.68
128	32	13.00	10.28	12.22	6.19	**5.03**
128	64	13.01	8.02	10.56	6.22	**5.05**
256	64	30.87	23.67	26.71	12.15	**9.67**
256	128	30.88	22.41	17.54	12.23	**9.73**

D WALL-CLOCK RUNTIME

Table 9 reports average generation wall-clock time in seconds per sample, measured on NVIDIA A40 GPUs. The unstarred TRACELOCK column is the measured end-to-end runtime of our current PyTorch/Transformers prototype. This prototype runs the controller outside the frozen D-LLM and explicitly materializes intermediate hidden states, which introduces extra GPU-memory traffic and framework overhead that would not necessarily remain in a fused implementation. In a production setting, the controller would ideally be integrated into the D-LLM decoding stack rather than invoked as an external module. In addition, our prototype uses a separate autoencoder projection stack as an engineering device for making hidden features tractable.

To measure the controller cost directly, we profile TRACELOCK and the native block-128 L2P filters on the same 100 fixed GSM8K samples, using generation length 256, window/block size 128, batch size one, BF16, and NVIDIA A40 GPUs. Table 8 separates end-to-end latency from the control-module cost. For TRACELOCK, this cost includes both the remasker and hidden-state projection; for L2P, it is the filter cost.

For TRACELOCK, the Dream profile consists of 0.066 s in the remasker and 0.085 s in the projection per sample; the corresponding LLaDA costs are 0.046 s and 0.117 s. This is approximately 2.0 ms of added control latency per decoding step on Dream and 3.8 ms on LLaDA, compared with 171.5 ms and 200.2 ms for the respective base D-LLM invocation. Peak VRAM rises from 15,570 to 16,225 MB on Dream ($+655$ MB, 4.2%) and from 15,600 to 16,212 MB on LLaDA ($+612$ MB, 3.9%). The TRACELOCK controller has 20.67M parameters, or approximately 0.27% of the 7.62B-parameter Dream backbone.

To separate this prototype overhead from the cost of the decoding policy itself, we also report TRACELOCK*, a corrected runtime estimate. For each backbone b and generation length L, we profile 100 samples with CUDA events and compute

$$\alpha_{b,L} = \frac{T_{\mathrm{DLM}}^{\mathrm{logits}} + T_{\mathrm{ctrl}}^{\mathrm{core}}}{T_{\mathrm{wall}}^{\mathrm{full}}}, \qquad T_{\mathrm{TL}^*} = \alpha_{b,L} T_{\mathrm{TL}}.$$

Here $T_{\mathrm{DLM}}^{\mathrm{logits}}$ is the GPU time of the frozen D-LLM forward without hidden-state extraction, $T_{\mathrm{ctrl}}^{\mathrm{core}}$ is the GPU time of the controller core, and $T_{\mathrm{wall}}^{\mathrm{full}}$ is the measured full prototype runtime. We estimate a separate $\alpha_{b,L}$ for LLaDA and Dream at each generation length, and apply the corresponding factor to the wall-clock values in the table. Thus TRACELOCK* should be interpreted as an implementation-corrected estimate, while the unstarred column remains the actual measured prototype runtime.

E Feature Details and Ablations

Detailed feature construction. TraceLock uses three late hidden-state snapshots from the frozen D-LLM together with two short-range hidden deltas. Let

$$h_{t,i}^{(1)}, h_{t,i}^{(2)}, h_{t,i}^{(3)} \in \mathbb{R}^d$$

denote the selected hidden states at position i and step t. We define

$$\Delta_{t,i}^{(1)} = h_{t,i}^{(2)} - h_{t,i}^{(1)}, \qquad \Delta_{t,i}^{(2)} = h_{t,i}^{(3)} - h_{t,i}^{(2)}.$$

The hidden states and deltas are compressed before controller training:

$$z_{t,i}^{(j)} = E_h(h_{t,i}^{(j)}), \qquad r_{t,i}^{(k)} = E_\Delta(\Delta_{t,i}^{(k)}).$$

Each token feature is a concatenation of the states,

$$q_{t,i} = \left[z_{t,i}^{(1)}; z_{t,i}^{(2)}; z_{t,i}^{(3)}; r_{t,i}^{(1)}; r_{t,i}^{(2)} \right],$$

followed by a learned projection into the controller space.

Feature ablations. Table 10 evaluates hidden-feature variants on held-out decoding traces, measuring future-stability prediction rather than final-answer quality directly.

Table 10: Feature ablations for future-stability prediction on held-out decoding traces. Metrics are computed on the test split. Best results are bolded.

Features	AUROC	AP	Acc@0.5
Last hidden only	0.914	0.926	0.854
Two deltas only	0.923	0.933	0.873
Three hidden states only	0.928	0.939	0.881
Three hidden states + two deltas	**0.936**	**0.945**	**0.888**

Using only the last hidden state already provides a strong stability signal, but it is weaker than feature sets that include multiple hidden snapshots or explicit short-range dynamics. The best results come from combining three hidden-state snapshots with two deltas.

F Deployment Mechanism Ablation

Table 11 gives the full results for the deployment ablation summarized in Figure 3.

In the "w/o soft-crop" ablation, TraceLock considers the full unresolved generation region instead of advancing a bounded local window from left to right. This removes the semi-autoregressive progression supplied by local-window decoding, which helps these frozen D-LLMs produce stable outputs. The resulting quality drop in Table 11 is therefore expected: this ablation changes the decoding schedule as well as the controller's decision scope, with the largest losses on LLaDA.

G Hidden-State Inputs for Blockwise Token Filtering

The larger-window results suggest that confidence-only token filters can become brittle when the block size increases. To isolate whether this behavior comes from the filter architecture or from the input representation, we run an additional diagnostic experiment that keeps the Learn2PD-style blockwise interface but replaces scalar confidence inputs with hidden-state features. The output interface is therefore the same as Learn2PD, but the input contains richer trace information.

Table 12 shows that replacing confidence with hidden-state features substantially improves the Learn2PD-style filter, especially on Dream. TraceLock remains stronger, indicating that representation choice is only part of the design: sequence-conditioned context, trace-state modeling, and adaptive deployment mechanisms also contribute.

Table 11: TRACELOCK deployment ablations across math, QA, and coding. The full model uses both soft-crop and dynamic thresholding. Math reports final-answer accuracy (%), QA reports average rank, and coding reports Pass@1 (%). Higher is better for math and coding; lower is better for QA.

Backbone	Gen len	Full	w/o soft-crop	w/o dynamic threshold	w/o both
LLaDA	128	**75.60**	56.5	68.7	56.2
LLaDA	256	**80.00**	58.0	70.5	55.5
Dream	128	**73.5**	70.0	71.5	69.1
Dream	256	**80.8**	77.4	77.8	76.6
LLaDA	128	**1.81**	2.96	2.18	3.06
LLaDA	256	**1.93**	2.92	2.16	2.99
Dream	128	**2.01**	2.81	2.27	2.92
Dream	256	**2.15**	2.69	2.39	2.77
LLaDA	128	**34.15**	27.4	31.1	23.8
LLaDA	256	**37.80**	28.0	33.5	26.8
Dream	128	**51.2**	47.6	49.4	45.7
Dream	256	**56.1**	50.0	49.4	48.8

Table 12: Hidden-state Learn2PD diagnostic on HumanEval at generation length 128 and block size 64. Results report Pass@1 (%). L2P-Hidden keeps the blockwise token-filter interface of Learn2PD, but replaces confidence-only inputs with concatenated hidden-state features. Higher is better.

Backbone	L2P-Confidence	L2P-Hidden	TRACELOCK
Dream	32.9	46.9	**51.2**
LLaDA	24.4	28.1	**32.32**

H OTHER VARIATIONS

H.1 DEPLOYMENT-AWARE SELF-TRAINING

A controller trained only on heuristic traces faces distribution shift at deployment time. Once TRACELOCK is inserted into the generation loop, it induces different partially completed sequences, different hidden states, and different future labels. We address this with deployment-aware self-training.

Let π_{θ_0} be a pretrained controller. We decode new prompts with π_{θ_0}, record the resulting traces, and again derive labels from the final sequence:

$$y^{\pi}_{t,i} = \mathbb{I}\left[\hat{x}^{\pi}_{t,i} = x^{\pi,\star}_{i}\right].$$

When collecting self-training traces, we use a slightly more conservative acceptance threshold so that the on-policy pool favors higher-quality completed trajectories. After filtering, these on-policy traces are mixed with the original trace pool:

$$\mathcal{D}_{\text{mix}} = \alpha\mathcal{D}_{\text{self}} + (1-\alpha)\mathcal{D}_{\text{pre}}.$$

The old traces are retained to avoid self-reinforcement bias, mode collapse, and forgetting of useful stability patterns learned from the original heuristic distribution. Table 13 shows that self-training mainly improves efficiency: TRACELOCK-ST executes fewer steps than the pretrained controller, while preserving comparable task quality in most settings. We hypothesize that this speedup comes from training on shorter on-policy traces, which teaches the controller to commit earlier under its own deployment distribution.

H.2 BEYOND SUPERVISED TRACE LEARNING: REINFORCEMENT LEARNING

Future-trace supervision is simple and stable, but it is still bounded by the quality of the traces used to train it. To explore whether the policy can move beyond supervised trace imitation, we

Table 13: TRACELOCK adaptation variants on LLaDA. Each cell reports the primary metric and average executed steps as metric / steps. Math uses accuracy, QA uses average rank, and coding uses Pass@1. Higher is better for math and coding; lower is better for QA.

Math (ACC / steps)

Setting	TRACELOCK	TRACELOCK-ST	TRACELOCK-RL
128/32	75.60 / 50.09	**75.8 / 47.6**	74.9 / 91.5
128/64	**74.0** / 50.58	73.3 / **42.4**	73.3 / 94.8
256/64	**80.00** / 85.78	78.9 / **87.9**	79.4 / 169.2
256/128	76.5 / 97.4	**77.1 / 88.9**	76.2 / 179.1

QA (rank / steps)

Setting	TRACELOCK	TRACELOCK-ST	TRACELOCK-RL
128/32	2.60 / 66.7	2.09 / **65.1**	**1.87** / 91.3
128/64	2.87 / 61.0	2.06 / **60.2**	**1.80** / 96.2
256/64	2.68 / 116.3	2.07 / **101.8**	**1.87** / 165.4
256/128	2.76 / 107.3	2.07 / **94.6**	**1.80** / 181.4

Coding (Pass@1 / steps)

Setting	TRACELOCK	TRACELOCK-ST	TRACELOCK-RL
128/32	**34.15 / 40.24**	29.9 / 44.4	32.3 / 77.9
128/64	**32.32 / 40.62**	31.1 / 47.6	31.1 / 80.7
256/64	**37.80** / 80.33	32.9 / **71.4**	36.6 / 133.0
256/128	**35.4** / 73.22	33.5 / **70.6**	33.5 / 139.5

also implement a reinforcement-learning extension. This component is not required for the main supervised method, but it gives a way to optimize the controller directly against final-answer rewards.

We view the frozen D-LLM and the current partial sequence as the environment. At step t, the policy samples token-level accept decisions from Bernoulli probabilities derived from TRACELOCK scores. To avoid sampling from extremely low-confidence positions, the implementation first applies a sample threshold and can rescale probabilities above that threshold:

$$\tilde{p}_{t,i} = \mathrm{clip}\left(\frac{p_{t,i} - \eta_{\text{sample}}}{1 - \eta_{\text{sample}}}, \epsilon, 1 - \epsilon\right).$$

The policy then samples $u_{t,i} \sim \mathrm{Bernoulli}(\tilde{p}_{t,i})$ for eligible positions.

After a full rollout, a reward model scores the completed answer. In our implementation this reward model is Skywork-Reward-V2 (Liu et al., 2025). For coding tasks, the reward is adjusted with executable checks. Let $\bar{R}$ be the mean reward in the current group before code-specific adjustment. If a generated code answer fails syntax checking, we subtract $|\bar{R}|$ from its reward; if it passes syntax checking but fails at runtime, we subtract $\frac{1}{2}|\bar{R}|$:

$$R_i^{\text{code}} = R_i - \begin{cases} |\bar{R}|, & \text{syntax error,} \\ \frac{1}{2}|\bar{R}|, & \text{runtime error,} \\ 0, & \text{otherwise.} \end{cases}$$

This prevents obviously invalid code from receiving high preference-model reward by accident, while penalizing syntax errors more strongly than runtime failures. Timeouts are treated as runtime failures.

We optimize the policy with group-relative advantages, following the spirit of GRPO-style training (Shao et al., 2024). For a group of G samples with rewards $\{R_j\}_{j=1}^{G}$, the advantage is

$$A_j = \frac{R_j - \mu_R}{\sigma_R + \epsilon}, \qquad \mu_R = \frac{1}{G}\sum_j R_j.$$

The policy loss for a trajectory is

$$\mathcal{L}_{\text{RL}} = -A_j \sum_t \log \pi_\theta(u_t \mid x_t) - \beta \sum_t \mathcal{H}(\pi_\theta(\cdot \mid x_t)).$$

The entropy term is important in practice: it keeps the token-level Bernoulli policy from collapsing too early, and provides a simple exploration pressure while the controller is still uncertain about which acceptance patterns lead to high final reward.

In practice, this RL stage is sensitive to both data quality and initialization. Training from pretrained or self-trained TRACELOCK checkpoints is substantially more stable than training the policy from scratch. Without a supervised initialization, the controller starts in a low-reward regime where high-quality trajectories are too sparse, making the reward signal weak and the policy difficult to bootstrap. We therefore treat RL as an extension on top of trace-supervised learning rather than as a standalone replacement: pretraining and self-training provide a strong warm start, and RL then refines the policy beyond what supervised traces alone can provide.

As shown in Table 13, RL refinement is most helpful on QA ranking, where it improves average rank across all listed settings. It usually uses more decoding steps, however, and gives mixed results on math and coding. We therefore treat it as a promising adaptation extension rather than part of the default TRACELOCK configuration.

Table 14: Trace divergence between confidence filtering and TRACELOCK on QA. Count metrics are normalized by the generation length of 128.

Step	Mask sym. diff (%)	Mask Jaccard (%)	Common-token diff (%)
16	36.8	54.1	1.0
32	33.6	51.1	3.9
48	30.6	46.9	8.3
64	25.2	45.4	17.1
80	21.0	40.7	21.1
96	13.1	33.4	37.2

Figures 5 and 6 summarize decoding dynamics for the three adaptation variants: the former reports the cumulative fraction of accepted tokens over steps, while the latter reports the average learned threshold over steps, with curves aggregated over randomly sampled examples from each task.

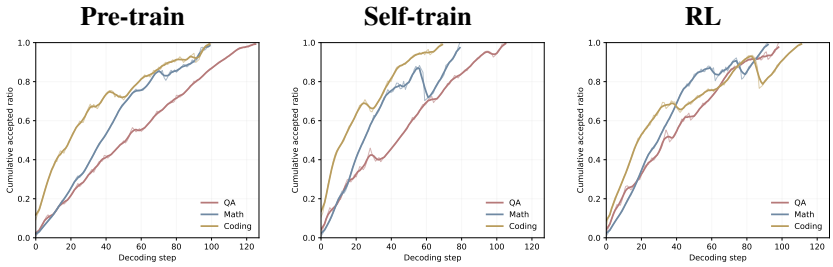


Figure 5: Cumulative acceptance ratio over decoding time on LLaDA with generation length 128. Left to right, the three panels show the pretrained controller, the self-trained controller, and the RL-refined controller.

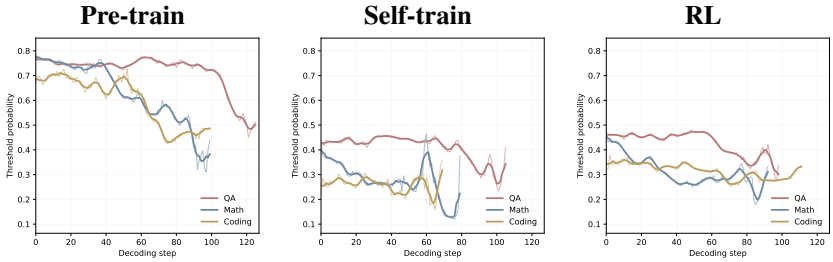


Figure 6: Hidden threshold trajectories over decoding time on LLaDA with generation length 128. The threshold evolves differently across tasks and controller variants, consistent with sequence-conditioned caution rather than a single global cutoff.

I CONFIDENCE AND TRACE DIAGNOSTICS

Divergence from confidence-based trajectories. We compare intermediate trajectories from confidence filtering and TRACELOCK on the same QA prompts. The diagnostic uses 100 prompts, generation length 128, block size 128, and probes both traces every 16 decoding steps. Table 14 reports mask symmetric difference, mask Jaccard overlap, and common-token difference among positions unmasked by both policies. The two policies differ substantially through the middle of decoding, and common-token differences increase at later steps. Figure 7 provides a complementary score-level view: TRACELOCK scores remain positively correlated with confidence, but the correlation changes over decoding time and is far from a fixed confidence ordering.

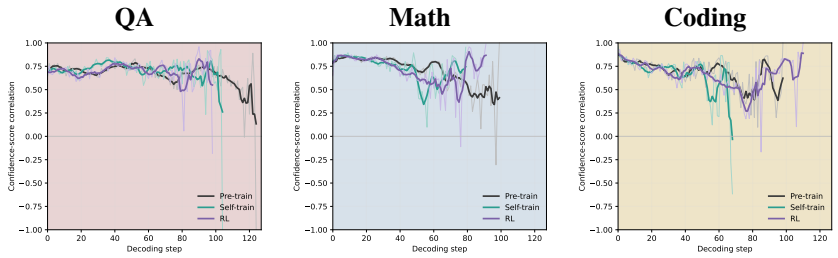


Figure 7: Stepwise correlation between base-model token confidence and TRACELOCK score on LLaDA with generation length 128. The learned stability score is positively related to confidence, but the correlation is imperfect and evolves during decoding.

Figures 8 and 9 visualize individual traces. Green cells indicate positions with identical token IDs between the two policies, while red cells indicate mismatched token IDs or a masked/unmasked disagreement.

J QA LLM-AS-JUDGE PROMPT

For QA evaluation, we use an LLM-as-judge protocol to compare candidate answers generated for the same question. The judge is instructed to score candidates only relative to the other candidates in the same batch, rather than against a global scale across all questions. The resulting scores are then converted into per-question rankings used in the QA evaluation tables. The prompt template is shown below.

```
System:
You are a strict scoring assistant. Return exactly one JSON object and
↪  nothing else.

User:
Score the candidate answers for the same question.

Rules:
1. Score candidates only relative to each other within this batch, not
↪  against a
   global scale across questions.
2. Use a 0 to 10 scale, where higher is better. Decimal scores are
↪  allowed.
3. Judge correctness first, then completeness, relevance, reasoning
↪  quality,
   factual accuracy, fluency, and clarity.
4. Use the following unified scoring rubric:
   - 9 to 10: The answer is strong and complete. It directly answers the
     question, is factually correct, well-structured, fluent, and covers
     ↪  the
     important aspects with appropriate depth.
   - 7 to 8: The answer is mostly correct and useful, but has some
   ↪  limitations,
     such as missing depth, incomplete coverage of important aspects,
     ↪  minor
     ambiguity, or weaker explanation.
   - 5 to 6: The answer is partially useful but has noticeable problems,
   ↪  such as
     factual mistakes, reasoning gaps, grammar or fluency issues, unclear
     wording, or missing key information.
   - 4 to 5: The answer has major problems. It may contain obvious
   ↪  errors, poor
     structure, substantial omissions, or frequent grammar/clarity
     ↪  issues, while
     still retaining some connection to the question.
```

```
   - 2 to 3: The answer is very poor. It may be truncated,
   ↪  self-repeating,
     incoherent, largely irrelevant, or unable to provide a usable
     ↪  response.
   - 0 to 1: The answer is almost unusable or malformed. It does not
   ↪  meaningfully
     answer the question, is nonsensical, empty, or fails to form a
     ↪  coherent
     answer.
5. Do not reward candidate ids, style, verbosity, or politeness by
↪  themselves.
6. Penalize hallucinated facts, contradictions, unsupported claims,
↪  severe
   repetition, truncation, and answers that fail to address the actual
   ↪  question.
7. The `scores` field must include every candidate exactly once, with no
↪  missing
   candidates and no extra names.
8. Equal scores are allowed when candidates are truly tied.
9. Output valid JSON only, matching this schema:
{
  "scores": {
    "candidate_1": 8.4,
    "candidate_2": 6.1
  },
  "reason": "short explanation"
}

Total candidates: {num_candidates}
Candidate ids that MUST all appear exactly once in `scores`:
{candidate_ids}

Question:
{question}

Candidates:
{candidate_answers}
```

K Experimental Details

Trace data. For each frozen backbone, we collect 8,000 completed decoding traces for controller training. The prompts are drawn from the same task families used in our experiments: mathematical reasoning, open-ended question answering, and code generation.

Each trace is converted into step-level supervision by comparing intermediate token proposals with the final completed sequence, following the future-stability target in Section 3.2. We hold out a validation split of trace samples for controller selection and diagnostic evaluation. Task-level evaluation uses the datasets and metrics described in Section 4.

Controller training. The main TraceLock controller uses compressed hidden-state features, a 3-layer Transformer encoder, 8 attention heads, hidden dimension 384, feed-forward dimension 768, dropout 0.1, position encoding, and a learned dynamic-threshold head.

We train with AdamW using learning rate 10^{-4}, weight decay 0.01, gradient clipping at 1.0, and random seed 42. Random prefix cropping is applied over the generated region during training to expose the controller to partial contexts similar to local-window deployment. Validation is run periodically on held-out trace samples, and the supervised controller checkpoint is selected by validation rollout-proxy $F_{0.5}$.

Compute resources. All experiments are run on NVIDIA A40 GPUs. Collecting the 8,000 supervised training traces for one frozen backbone takes roughly two GPU-hours. Training the main TraceLock controller runs for about 10,000 optimization steps and takes approximately 20 minutes on one A40. For the optional adaptation stages, self-training collects about 1,000 on-policy traces and

then trains for 1,000 steps, taking about 5 minutes for the training stage. The reinforcement learning extension uses group size 4 and trains for 1,000 steps, taking about 5 GPU-hours.

Prompt: Explain the term "AI explainability".

Step	Confidence filtering	TraceLock
Final	AI explainability refers to the ability of artificial intelligence systems to provide clear and understandable explanations for their decisions and actions. This is important because AI systems are often based on complex algorithms and data, which can be difficult to understand and interpret. AI explainability aims to make AI systems more transparent and interpretable, so that users can understand how the system arrived at a particular decision or recommendation. This can help build trust in AI systems and make them more suitable for use in critical applications, such as healthcare, finance, and law. AI explainability can be achieved through various techniques, including feature attribution, model interpretation, and decision visualization.	AI explainability refers to the ability of artificial intelligence (AI) systems to provide explanations for their decisions that are interpretable and understandable to humans. This means that the AI system is able to communicate its decision-making process, reasoning, and decision to users. The goal of AI explainability is to to improve the reliability and fairness of AI systems, and to build trust, transparency, and accountability in AI applications. By providing explanations for AI decisions, AI explainability can help users understand how decisions are made and and how informed them, and it can allow for feedback and adjustments to improve the accuracy and fairness of AI systems.
32	AI explain ability refers to the ability of artificial intelligence systems to provide clear and understandable explanations for their decisions and actions . This is important because AI systems are often based on complex ∅ ∅ ∅

Prompt: Generate a dialogue between a customer and a salesperson in a department store.

Step	Confidence filtering	TraceLock
Final	Customer: Hi, I'm looking for a new dress for an upcoming event. Can you help me find one? Salesperson: Of course! Are you looking for a specific style or color? Customer: I want something elegant and sophisticated, preferably in a dark color. Salesperson: Okay, let me check our inventory. Here are a few options that fit your description. Would you like to try them on? Customer: Yes, please. Salesperson: Here you go. Customer: This is perfect! Thank you. Salesperson: You're welcome! We hope you enjoy the dress.	Customer: Hi, I'm looking for a new dress for my upcoming wedding. Can you help me find something? Salesperson: Of course! We have a wide variety of dresses to choose from. Do you have any specific style or color in mind? Customer: I'm looking for something simple and elegant, preferably in a color that complements my skin. Salesperson: Okay, I see. We have a few options that fit that description. Let me show you some examples. Here's a light blue dress that would complement your skin, and here's a simple white dress with a subtle print.
32	Customer : Hi , I 'm looking for a new dress for an upcoming event . Can you help me find one ? ⟷ Sales person : Of course ! ∅ you ∅ ∅ ∅