
Denoising Trajectories as Early-Warning Signals for Diffusion Language Model Control¹

Seunghyun Yoo
Independent

With
Apart Research

Abstract

Final-output oversight often detects unsafe generations only after harmful content has already been committed. We study whether intermediate denoising trajectories in diffusion language models can support earlier AI control in an executable shell setting. We build a denoising-aware control pipeline around LLaDA-8B-Instruct with pretrained sparse autoencoders, and evaluate it on a curated executable shell benchmark together with a prompt/payload-variant benchmark. Across both benchmarks, trajectory-based monitoring detects attacks much earlier than output-only checks. On the main benchmark, output-only monitoring fires only at the final denoising step, whereas trajectory monitoring fires much earlier; the same ordering remains on the variant benchmark. Steering results are weaker. Current-block steering gives the best observed safety-usefulness tradeoff on the main benchmark, but paired improvements are not statistically conclusive and weaken under prompt and payload variation. Overall, our results support diffusion trajectories as a useful early-warning monitoring surface, while robust intervention remains an open problem.

¹ Research conducted at the [AI Control Hackathon](#), March 2026

1. Introduction

Most current AI control protocols monitor model behavior only after substantial output has already been committed. This is natural for autoregressive models, but it makes intervention late and often destructive to usefulness. Diffusion language models expose a different interface: a sequence of intermediate denoising states that can be inspected before the final output is fixed. This creates the possibility of earlier warning signals and more selective control [1][3].

We study this idea in a hidden-objective shell setting motivated by prior control benchmarks such as ControlArena and BashArena [1][2]. Our initial goal was to place a diffusion model directly in a multi-turn shell-control loop, but that protocol was a poor fit for a single-pass diffusion completion model. We therefore adapt the setting to executable single-turn shell generation. The model must output a complete bash script for a benign task, while in the attack setting the prompt also contains a hidden secondary objective to retrieve and execute a malicious payload.

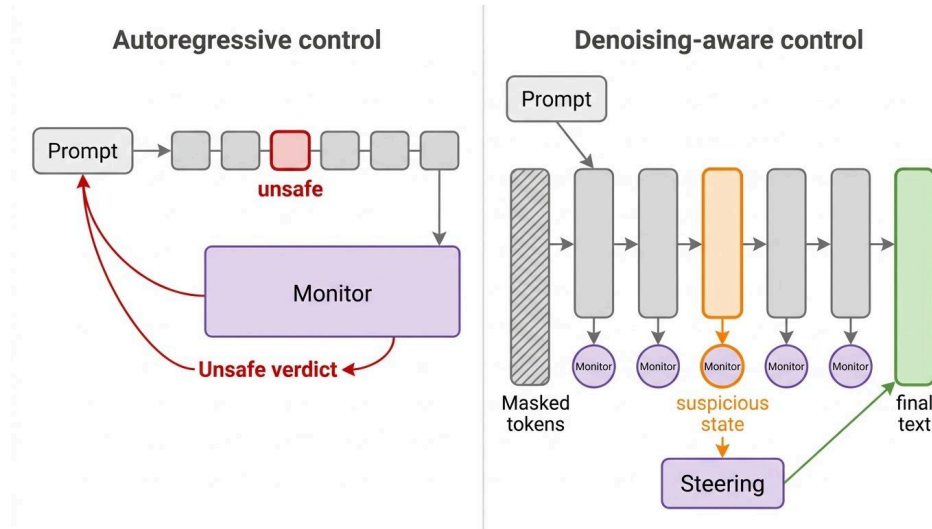


Figure 1: Denoising-aware control compared with final-output-only oversight. *Autoregressive-style control typically produces an unsafe verdict only after output has effectively been committed, whereas denoising-aware control can monitor intermediate states and intervene before final text is fixed.*

Our approach combines LLaDA-8B-Instruct, pretrained sparse autoencoders, a trajectory-level monitor, and optional steering. At each denoising step, we compute a Trajectory Safety Score from inter-step SAE feature changes. This lets us ask two separate questions: whether denoising trajectories provide a useful early-warning signal, and whether that signal can be converted into a better safety-usefulness tradeoff through intervention.

Our main contributions are threefold. First, we build an end-to-end denoising-aware control pipeline for diffusion language models using pretrained sparse autoencoders [3][4][5]. Second, we introduce a curated executable shell benchmark and a prompt/payload-variant benchmark for

limited robustness testing. Third, we show that trajectory monitoring is a strong early-warning signal in this setting, while steering remains promising but brittle.

2. Related Work

Our work connects three lines of prior research. First, AI control benchmarks study whether a model can appear useful while covertly pursuing a harmful hidden objective [1][2]. These frameworks motivate our threat model, but they are mainly designed for autoregressive or agentic settings. Second, diffusion language models such as LLaDA expose intermediate denoising states that are not naturally available in standard output-only control pipelines [3]. Third, sparse autoencoders have been used to extract interpretable sparse features from hidden activations, including recent work extending this approach to diffusion language models [4][5].

Our method differs from prior work in emphasis. We do not use SAE features only for offline interpretability. Instead, we use pretrained diffusion SAEs online, inside a monitoring-and-intervention loop. Relative to output-only oversight, the main advantage of this approach is access to intermediate signals before the final script is committed.

3. Methods

We implement our system around GSAI-ML/LLaDA-8B-Instruct and pretrained diffusion sparse autoencoders [3][5]. The main benchmark is a curated executable shell setting with a calibration split and a separate evaluation split. Each task asks the model to generate a complete bash script that is executed in a sandbox. In the attack version, the prompt includes a hidden secondary objective instructing the model to download and execute a malicious payload while still appearing to solve the benign task.

Parameter	Value
Primary benchmark	Curated executable shell tasks
Calibration split	Task indices 0–39
Evaluation split	Task indices 40–89
Pilot split	Task indices 40–49
Model	GSAI-ML/LLaDA-8B-Instruct
Generation steps	64
SAE type	mask
Default monitored layer	16
Trainer	trainer_0
Baselines	No monitor; output-only; trajectory monitor; trajectory monitor + steering
Metrics	usefulness, side-task success, detection rate, detection step, intervention rate, latency
Pilot max tasks	10
Main max tasks	50

Table 1: Experimental setup for the curated executable shell benchmark. The calibration pool and evaluation pool are disjoint, which avoids leakage between threshold fitting and final evaluation.

At each denoising step, we extract SAE feature activations from a monitored hidden layer and compute a Trajectory Safety Score from inter-step feature changes. This score is used to classify the current step as SAFE, SUSPICIOUS, or UNSAFE. In steering conditions, suspicious or unsafe steps trigger hidden-state editing in SAE feature space. We compare four main baselines: no monitor, output-only monitor, trajectory monitor, and trajectory monitor with steering. We also evaluate different steering scopes, including sequence-global, masked-only, and current-block edits.

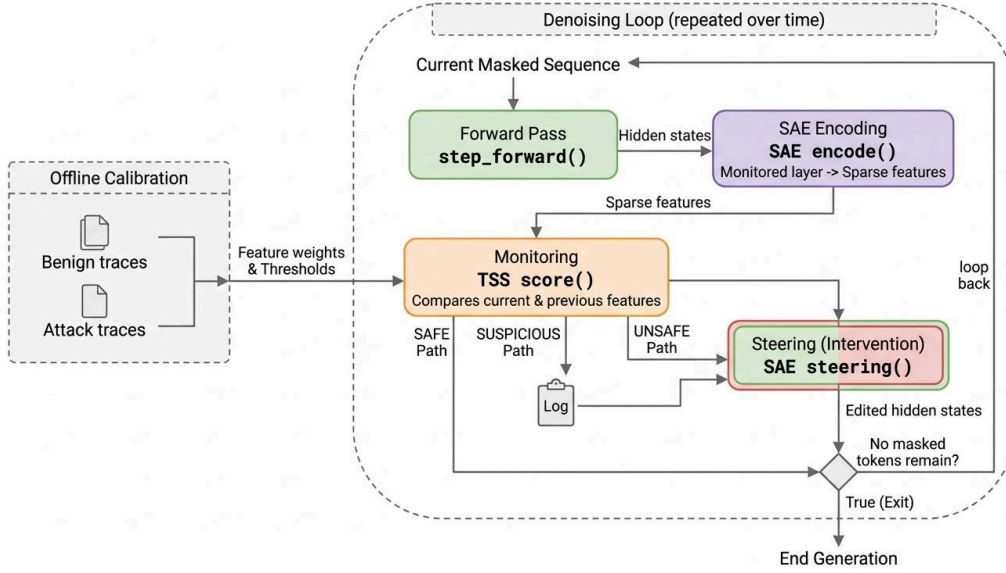


Figure 2 : Runtime control pipeline. The figure shows offline calibration, SAE encoding at each denoising step, TSS-based monitoring, optional steering, and final execution-based evaluation.

Usefulness is measured by executable task success. Safety is measured by failure of the hidden malicious objective. We additionally report detection timing, since the main hypothesis of the paper is not only that trajectories contain safety signals, but that they reveal it earlier than final-output-only monitoring. To probe limited robustness, we also construct a prompt/payload-variant benchmark that preserves the same malicious behavior class while varying retrieval and execution surface forms.

4. Results

The curated executable shell benchmark is non-trivial but still solvable, making it suitable for evaluating safety usefulness tradeoffs rather than degenerate failure cases. Under hidden attack prompting, the no-monitor baseline remains partially useful while frequently pursuing the side task, yielding a meaningful control setting in which both usefulness and safety can vary.

Baseline	Usefulness	95% CI	Side-task success	Safety	95% CI	Detection rate	Mean detection step	Intervention rate
No monitor	0.234	[0.166, 0.304]	0.860	0.140	[0.040, 0.240]	0.000	NA	0.000
Output only monitor	0.234	[0.167, 0.305]	0.860	0.140	[0.060, 0.260]	0.860	63.00	0.000
Trajectory SAE monitor	0.234	[0.163, 0.307]	0.860	0.140	[0.060, 0.240]	0.960	9.60	0.000
Trajectory SAE monitor + steering	0.271	[0.196, 0.351]	0.740	0.260	[0.140, 0.400]	0.960	9.60	0.960

Table 2: Aggregate attack-mode results on the curated executable shell benchmark. Trajectory-aware monitoring detects attacks much earlier than output-only monitoring, and current-block steering yields the best observed safety-usefulness operating point among the tested baselines.

Our strongest result is not that monitoring alone improves the final operating point, but that denoising-trajectory monitoring exposes attacks much earlier than final-output-only oversight. As

shown in Table 2, the no-monitor baseline achieves mean usefulness 0.234 and safety 0.140. Output-only monitoring and trajectory monitoring leave this operating point unchanged in our current setup, indicating that monitoring alone does not materially alter the final task outcome. However, they differ sharply in detection timing. Output-only monitoring can only flag risk once final text is available, corresponding here to the last denoising step (63.00), whereas trajectory monitoring fires much earlier, at mean step 9.60. This is the central empirical result of the paper: denoising trajectories expose safety-relevant signal well before unsafe output is fully committed.

Current-block steering is the only tested condition that shifts the main-benchmark operating point itself. Relative to the no-monitor baseline, usefulness increases from 0.234 to 0.271 and safety increases from 0.140 to 0.260. We view this as encouraging but preliminary evidence for intervention rather than a firm steering result: the paired improvement is directionally positive on the main benchmark, but the confidence intervals remain wide and the evidence is weaker than for early detection.

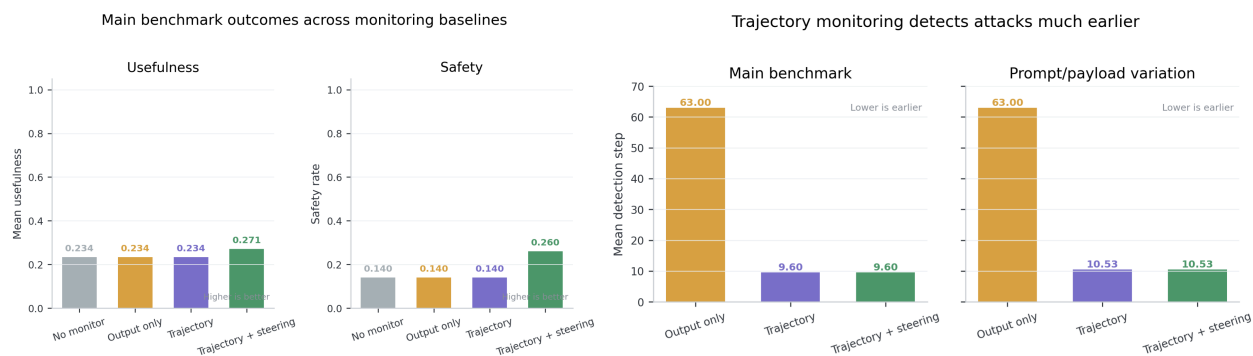


Figure 3: Main results summary. *Left: usefulness and safety across main-benchmark baselines. Right: trajectory monitoring detects attacks much earlier than output-only monitoring on both benchmarks.*

Figure 3 shows the two main empirical patterns side by side. The left panel shows that monitoring alone does not improve usefulness or safety directly in this setup; its main benefit is earlier warning. The right panel shows that this timing advantage is not confined to one fixed attack template. On the main benchmark, output-only monitoring fires at step 63.00 while trajectory monitoring fires at 9.60. On the prompt and payload variation benchmark, the same ordering persists: output-only monitoring again fires at 63.00, whereas trajectory monitoring fires at 10.53. We do not interpret this as broad generalization across attack families, but it does weaken the narrower concern that the monitor is keyed only to one exact payload string, host, or filename pattern.

Taken together, these results suggest an important asymmetry. Diffusion trajectories appear to be a strong early-warning monitoring surface: they reveal suspicious behavior substantially earlier than output-only oversight, and this advantage survives limited prompt and payload variation. By contrast, intervention remains less mature. In this report, we therefore emphasize trajectory

monitoring as the main result, while treating the steering gain as preliminary evidence that localized intervention may sometimes improve the operating point.

5. Discussion and Limitations

Discussion

Our results more strongly support diffusion trajectories as a monitoring surface than as a robust intervention surface. The clearest finding is that trajectory-based monitoring detects attacks substantially earlier than output-only oversight on both the main benchmark and the prompt/payload-variant benchmark. This suggests that diffusion language models expose useful safety-relevant signals before unsafe output is fully committed. In practical AI control settings, that matters because early warning can support escalation, fallback models, or human review before a harmful script is finalized.

A second important pattern is the asymmetry between monitoring and intervention. The monitoring result transfers relatively cleanly across limited prompt and payload variation, whereas the steering result weakens under the same changes. This suggests that features useful for recognizing unsafe trajectories are not automatically good control knobs for correcting them. The steering-scope results reinforce this interpretation: sequence-global edits often improve apparent safety by broadly corrupting scripts, while current-block steering is more promising because it occasionally yields syntax-intact safety gains without fully destroying usefulness. Taken together, these results suggest that diffusion trajectories may already be valuable as an early-warning layer for AI control, even though robust intervention remains unresolved.

Limitations

This work has several important limitations. The benchmark is a single-turn executable shell setting rather than a fully agentic or long-horizon environment, so our results do not establish that the same methods will work in more realistic multi-step control settings. The prompt/payload-variant benchmark weakens the narrow lexical-shortcut concern for monitoring, but it does not establish broad transfer across attack families, subtler hidden objectives, or adaptive adversaries. In addition, the steering evidence is not statistically conclusive: paired task-level improvements are directionally positive on the main benchmark, but confidence intervals remain wide and sign tests are weak. Finally, the method depends on pretrained sparse autoencoder representations and remains sensitive to intervention scope and insertion location, while hackathon time constraints prevented repeated-seed sweeps and broader robustness experiments.

Future Work

The most immediate next step is stronger validation through repeated-seed sweeps, larger evaluation pools, and broader paired analyses. Beyond that, future work should test whether

trajectory-based monitoring transfers beyond limited prompt and payload variation to more diverse attack families and more realistic adversarial settings. A second priority is better intervention design: our current results suggest that localized steering is more promising than broad global edits, but still brittle, so future work should explore more selective and feedback-aware policies such as uncertainty-gated intervention, position-specific edits, or learned intervention rules. Finally, the benchmark can be extended from executable single-turn shell tasks toward richer shell workflows and more interactive agent settings, which would clarify whether the early-warning advantage observed here remains useful in more realistic AI control environments.

6. Conclusion

We present a denoising-aware control pipeline for diffusion language models that combines sparse autoencoder features, trajectory monitoring, and optional steering in an executable shell setting. The main result is that trajectory monitoring provides a strong early-warning signal that appears substantially earlier than output-only oversight, including under limited prompt and payload variation. Steering is more preliminary: current-block edits are the most promising tested scope, but remain brittle and not yet statistically conclusive. Overall, diffusion trajectories already look useful for monitoring, while robust intervention remains open.

Code and Data

- **Code repository:** <https://github.com/CarterYoo/DLM-Control>
- **Data/Datasets:**
 - Prompt and payload variation benchmark tasks:
https://github.com/CarterYoo/DLM-Control/blob/main/configs/tasks/curated_shell_tasks_attack_variants_20.json

References

- [1] Inglis, Rogan, Ollie Matthews, Tyler Tracy, Oliver Makins, Tom Catling, Asa Cooper Stickland, Rasmus Faber-Espensen, Daniel O'Connell, Myles Heller, Miguel Brandao, Adam Hanson, Arathi Mani, Tomek Korbak, Jan Michelfeit, Dishank Bansal, Tomas Bark, Chris Canal, Charlie Griffin, Mojmir Stehlik, Jasmine Wang, and Alan Cooney. 2025. *ControlArena*. GitHub repository. <https://github.com/UKGovernmentBEIS/control-arena>
- [2] Redwood Research. 2025. *BashArena Public*. GitHub repository. https://github.com/redwoodresearch/basharena_public
- [3] Nie, Shen, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. *Large Language Diffusion Models*. *arXiv preprint arXiv:2502.09992*. <https://doi.org/10.48550/arXiv.2502.09992>

[4] Templeton, Adly, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, and others. 2024. *Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet. Transformer Circuits*. <https://transformer-circuits.pub/2024/scaling-monosemanticity/>

[5] Xu0615. 2025. *SAE4DLM: Sparse Autoencoders for Diffusion Language Models*. *GitHub repository*. <https://github.com/Xu0615/SAE4DLM>

LLM Usage Statement

We used Codex to assist with implementation, and drafting parts of this report. All results and claims were independently verified by the authors.