Knowledge Editing in Masked Diffusion Language Models

Anonymous ACL submission

Abstract

Knowledge editing aims to update or correct factual knowledge in a language model. A widely used approach, *locate-then-edit*, does this in two steps: it first localizes a fact within the model, then edits the weights there. To date, such methods have been developed exclusively on autoregressive models (ARMs). Whether their underlying assumptions hold for masked diffusion models (MDMs), which model text bidirectionally and generate by iterative denoising rather than next-token prediction, remains an open question. We address it by transferring locate-then-edit to MDMs and comparing two MDMs (LLaDA, Dream) with two ARMs (LLaMA, Qwen) at matched scale. Our central finding has two parts. First, *where* an edit is applied transfers across paradigms: causal tracing highlights the same early-to-mid-layer MLP at the last subject token in both, and editing is most effective there. Second, this shared location does not guarantee a shared outcome. Single-token edits succeed in both, but as targets grow longer, editing degrades systematically in the MDMs but not the ARMs. The failure stems from *how* the edited fact is generated: producing a multi-token target requires passing through partially unmasked intermediate states for which the edit was never optimized. Guided by this diagnosis, we introduce a simple correction that optimizes the edit for these states, substantially restoring multi-token performance.[1]

1 Introduction

Large language models (LLMs) acquire extensive factual knowledge through pretraining, but not all of it is correct: some facts are simply wrong, a product of noise in the training data, and others grow stale over time (Ji et al., 2023; Lin et al., 2022; Jang et al., 2022). Fine-tuning can fix them, but doing so over the full model is computationally expensive and risks overfitting to the correction while eroding unrelated knowledge through catastrophic forgetting (Bethune et al., 2025; Luo et al., 2025; Zhu et al., 2020). Knowledge editing offers an alternative, revising targeted facts while keeping the rest of the model's knowledge intact (Yao et al., 2023; Wang et al., 2024b). A fact is typically represented as a tuple (s, r, o)—a subject s, a relation r, and an object o—and an edit replaces the object with a new *target object* o^* for a given (s, r): for example, "Lionel Messi (s) plays for (r) FC Barcelona (o)" can be edited to "Inter Miami" (o^*). Among such methods, the *locate-then-edit* approach identifies where a fact is causally localized by analyzing the model's internals, then modifies only the weights at that location (Meng et al., 2023a,b). Because the edit is grounded in an explicit analysis rather than distributed across the whole model or delegated to an auxiliary network (De Cao et al., 2021; Mitchell et al., 2022a,b), the edit site is interpretable.

These methods, however, were developed and validated entirely on autoregressive models (ARMs). Masked diffusion models (MDMs) (Nie et al., 2025; Ye et al., 2025; Zhu et al., 2025) have recently emerged as a competitive alternative paradigm, yet knowledge editing for MDMs remains largely unexplored. Whether locate-then-edit, designed for ARMs, also applies to MDMs is the question this work takes up. The challenge runs deeper than reimplementation: the location these methods modify is not arbitrary but derived from how ARMs work. Using causal tracing, ROME (Meng et al., 2023a) localized a fact's recall to the early-to-mid-layer MLP of the last subject token, read this MLP as a form of associative memory, and made it the edit target; MEMIT (Meng et al., 2023b) built on the same site to scale to many edits at once. This grounding, however, comes entirely from ARMs, which differ from MDMs even in how knowledge is acquired: an ARM is trained to predict the next token, whereas an MDM recov-

[1] We will release our code upon publication.

ers masked tokens from bidirectional context. It is therefore not obvious that an edit site identified in ARMs carries over to a model trained so differently.

To investigate this, we transfer locate-then-edit to MDMs without altering its algorithm, applying only the adjustments needed for the MDM's masked prediction (§3.3), and we compare ARMs (LLaMA, Qwen) and MDMs (LLaDA, Dream) under matched scale. We examine the question at two levels: we perform causal tracing to locate where facts are causally localized, and we measure editing performance across token positions and layer ranges to identify where editing is most effective. We find that even in MDMs, where every token attends to the full subject bidirectionally, the localization of facts coincides with that in ARMs—the early-to-mid-layer MLP of the last subject token. The location assumption underlying locate-then-edit is thus not specific to autoregressive training but holds across both paradigms.

Sharing a location, however, does not mean sharing an outcome. While the two paradigms perform comparably when the target object is a single token, the MDM degrades substantially once the target spans multiple tokens. The cause lies not in the location but in the generation process. Locate-then-edit fits the edit on a single state. An ARM passes through exactly that state as it decodes left to right, so the edit operates under the conditions it was fit for. An MDM, by contrast, starts from a fully masked output and unmasks its positions over several denoising steps, passing through a chain of partial-mask states the edit was never optimized for. As a result, editing degrades systematically as the target spans more tokens. Building on this diagnosis, we propose a simple correction that makes the edit robust to these intermediate states, and show that it substantially restores multi-token editing in MDMs.

Our contributions are as follows:

- We present the **first analysis of locate-then-edit knowledge editing in MDMs** (§3), showing that its core location assumption—that recall is mediated by the early-to-mid-layer MLP at the last subject token—holds across both paradigms.

- We find that in MDMs, locate-then-edit suffers a **multi-token breakdown** (§4): editing succeeds for single-token target objects but degrades systematically as the target object grows longer, which we trace to a mismatch between single-location editing and iterative denoising. Standard benchmarks, dominated by single-token targets, cannot reveal this, so we surface and analyze the breakdown with a multi-token evaluation adapted from KAMEL (Kalo and Fichtel, 2022).

- We propose a **simple correction** (§4.3) that makes the edit robust to the partial-mask states traversed during generation, substantially restoring multi-token editing performance in MDMs.

2 Preliminaries

2.1 Locate-then-Edit Knowledge Editing

Knowledge editing aims to modify a fact triplet from (s, r, o) to (s, r, o^*). Locate-then-edit methods (Meng et al., 2023a,b) achieve this by viewing the down-projection layer $\mathbf{W}$ of a Transformer MLP as a linear associative memory (Anderson, 1972; Kohonen, 1972) that maps a key vector $\mathbf{k}$ to a value vector $\mathbf{v}$ via $\mathbf{W}\mathbf{k} = \mathbf{v}$. The key $\mathbf{k}$, taken as the MLP up-projection activation at the last token of the subject s, encodes the entity, while the value $\mathbf{v}$ encodes the associated (r, o). Editing a fact amounts to adding an update $\mathbf{\Delta}$ to $\mathbf{W}$ so that the same key maps to a new value $\mathbf{v}^*$ encoding (r, o^*), i.e., $(\mathbf{W} + \mathbf{\Delta})\mathbf{k} \approx \mathbf{v}^*$. This choice of where to edit rests on *causal tracing* (Meng et al., 2023a), which identifies the components most responsible for recalling a fact—the early-to-mid-layer MLPs at the last subject token.

Building on this site, MEMIT (Meng et al., 2023b) edits a contiguous range of these MLP layers and, for a batch of facts, solves for the update $\mathbf{\Delta}$ in closed form from the keys $\mathbf{k}$ and the corresponding target values $\mathbf{v}^*$ (full derivation in Appendix A). While the key $\mathbf{k}$ is read directly from the subject, the target value $\mathbf{v}^*$ is obtained by optimization: it is the value that, when written at the edit location, drives the model to produce the target object o^* given the prompt p describing (s, r). Concretely, $\mathbf{v}^* = \mathbf{v} + \boldsymbol{\delta}$, where the residual $\boldsymbol{\delta}$ is found by gradient descent to maximize the probability of o^* under the model with the edit-site value replaced by $\mathbf{v}+\boldsymbol{\delta}$, denoted $G(\mathbf{v} \leftarrow \mathbf{v} + \boldsymbol{\delta})$:

$$\boldsymbol{\delta} = \arg\min_{\tilde{\boldsymbol{\delta}}} \; -\log \mathbb{P}_{G(\mathbf{v} \leftarrow \mathbf{v}+\tilde{\boldsymbol{\delta}})}\big[\, o^* \mid p \,\big]. \quad (1)$$

The optimization details follow MEMIT (Meng et al., 2023b) (Appendix A). Crucially, $\boldsymbol{\delta}$ is fit

for a single conditioning state. This implicitly assumes the model encounters that same state when it generates the target—an assumption we revisit for MDMs in §4.

2.2 Masked Diffusion Language Models

ARMs factorize the joint distribution into left-to-right conditionals, predicting each token from its left context only through a causal attention mask. An MDM is trained differently. Given a sequence $x_0 = (x_0^1, \ldots, x_0^L)$, a forward process independently replaces each token with an absorbing mask state M with probability $t \in [0, 1]$, and the model learns a *mask predictor* $p_\theta(\cdot \mid x_t)$ that recovers all masked tokens simultaneously from the partially masked sequence x_t. Training minimizes a cross-entropy loss evaluated only at masked positions (Nie et al., 2025):

$$\mathcal{L}(\theta) = -\,\mathbb{E}_{t,\,x_0,\,x_t}\Bigg[\frac{1}{t}\sum_{i=1}^{L}\mathbf{1}\big[x_t^i = \mathrm{M}\big] \log p_\theta\big(x_0^i \mid x_t\big)\Bigg], \quad t \sim \mathcal{U}[0,1]. \tag{2}$$

Two properties of this formulation are central to our study. First, the mask predictor attends to the *entire* input without a causal mask, so every position is predicted from bidirectional context rather than from a left prefix. Second, the model is trained to reconstruct masked tokens rather than to predict the next one, so the factual knowledge that locate-then-edit modifies is formed under a learning objective different from that of ARMs. We study two MDMs that realize this objective differently: LLaDA is trained from scratch, whereas Dream is initialized from the weights of an ARM (Qwen2.5) and trained with a shifted objective that aligns its prediction with that of an ARM (Ye et al., 2025).

Iterative denoising. Unlike AR decoding, which produces tokens one at a time, an MDM generates by *iterative denoising*. Starting from a fully masked output of length L, at each step the model predicts all masked positions, unmasks a subset of them (e.g., the most confident ones), and leaves the rest masked for subsequent steps (Nie et al., 2025). Producing a multi-token answer therefore unfolds over several steps and necessarily passes through *intermediate states* in which some positions are already filled while others remain masked. As we show in §4, these intermediate states—which have no counterpart in AR generation—are where an edit optimized for a single conditioning state can fail to take effect.

3 What Transfers: Locating and Editing Facts

The locate-then-edit approach proceeds in two stages: causal tracing identifies the components most responsible for recalling a fact, and the weights at those locations are then modified. We test whether each stage transfers to MDMs. After the setup and our MDM adaptation (§3.1), we apply causal tracing to locate *where facts are causally localized* (§3.2) and measure *where editing is most effective* across token positions and layers (§3.3). In both, despite the MDM's bidirectional training, it localizes facts to the same site as the ARMs—the early-to-mid-layer MLP of the last subject token.

3.1 Setup

Models. We study two ARM–MDM pairs matched in parameter count and layer depth, so each comparison controls for model capacity: LLaMA-3-8B-Instruct (Grattafiori et al., 2024) (ARM) / LLaDA-8B-Instruct (Nie et al., 2025) (MDM), and Qwen2.5-7B-Instruct (Qwen et al., 2025) (ARM) / Dream-v0-Instruct-7B (Ye et al., 2025) (MDM). The latter is especially controlled, as Dream is initialized from the Qwen2.5 weights.

Dataset and metrics. We evaluate on COUNTERFACT (Meng et al., 2023a), the standard knowledge-editing benchmark, using its three criteria (Meng et al., 2023a,b): *efficacy* (the rewrite prompt (s, r) elicits the new target), *generalization* (paraphrase prompts elicit it too), and *specificity* (a neighborhood prompt, with a distinct subject s' sharing r and o, still yields the original object o). Following MEMIT's batched setting, we edit 500 facts in a single batch. We assess all three from the model's generation; details are deferred to Appendix B.

3.2 Locating Facts via Causal Tracing

Causal tracing. To localize where a fact is recalled, we follow the causal tracing procedure of Meng et al. (2023a). Since we ask where the *subject* information is processed en route to the prediction, we corrupt the subject tokens and then measure how much restoring an individual hidden state to its clean value recovers the correct prediction. For each fact, we compare three runs:

- **Clean run.** We pass the unperturbed input x through the model and collect all hidden states

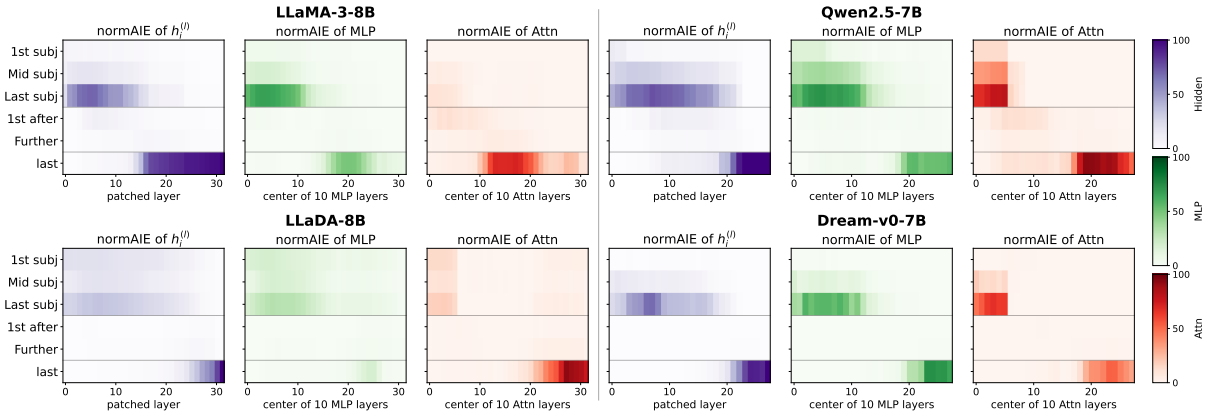

Figure 1: Causal tracing across ARM/MDM pairs. Top row: ARM baselines; bottom row: their MDM counterparts. 'last' refers to the final input token (in MDMs, the last appended mask token).

$\{h_i^{(l)}\}$; the model produces the correct answer o.

- **Corrupted run.** To obfuscate the subject, we add Gaussian noise $\epsilon \sim \mathcal{N}(0, \nu^2)$ to the embeddings of all tokens spanning the subject entity, with $\nu = 3\sigma$, where σ is the standard deviation of subject token embeddings, estimated once per model. The model then runs normally and, having lost subject information, typically predicts incorrectly.
- **Corrupted-with-restoration run.** We proceed with the corrupted input but restore clean states at a chosen $(\hat{i}, \hat{l})$. Writing each layer's update as $h_i^{(l)} = h_i^{(l-1)} + a_i^{(l)} + m_i^{(l)}$, we vary the restored mediator: the full hidden state $h_{\hat{i}}^{(\hat{l})}$ when tracing hidden states, or only the MLP contribution $m_{\hat{i}}^{(\hat{l})}$ or attention contribution $a_{\hat{i}}^{(\hat{l})}$ when tracing sub-modules. Since these per-layer contributions are small, restoring one layer alone has little effect, so for sub-modules we restore a contiguous window of layers around $\hat{l}$ at token $\hat{i}$, following Meng et al. (2023a).

The *indirect effect* (IE) of a mediator is the increase in the probability of o when it is restored, relative to the corrupted run, and averaging over facts yields the *average indirect effect* (AIE). For comparability across models, we report a *normalized AIE*: the recovered probability gain, $P_{\text{restored}}(o) - P_{\text{corrupted}}(o)$, as a fraction of the full corruption-induced gap, $P_{\text{clean}}(o) - P_{\text{corrupted}}(o)$, expressed as a percentage and averaged over facts. A value of 0 means the restored state contributes nothing beyond the corrupted run, and 100% means it fully recovers the clean prediction. Our interest is in *where along the layers*, and in *which sub-module* (MLP vs. attention), the effect concentrates. We apply the same procedure to all four models. For the two MDMs, the model predicts at mask positions rather than at the next token, so we append a block of mask tokens to the prompt and read the score $P(o)$ at the first mask position, where the first token of o is predicted. We give the noise level, restoration window size, mask block length, MDM adaptation, and remaining tracing details in Appendix C.

Results. Figure 1 decomposes the normalized AIE over token positions and layers for all four models. In every model, both ARMs and MDMs, the two causal sites reported by Meng et al. (2023a) emerge: an *early site* at the last subject token in the early-to-mid layers, carried mainly by the MLP, and a *late site* at the answer position in the upper layers, dominated by attention. The late site is expected, since the model reads out its prediction there; the early site, which Meng et al. (2023a) interpret as the site of factual recall, is our focus.

Crucially, this early site holds for the MDMs as well. Despite being trained on bidirectional context, both LLaDA and Dream localize factual recall to the same early-to-mid-layer MLP at the last subject token found in the ARMs.

LLaDA's early-site signal is visibly weaker than the ARMs', but this reflects the distribution of $P_{\text{clean}}(o)$ rather than weaker localization: LLaDA assigns low probability to the correct answer on more facts, and such facts yield weak peaks that pull down the average. Stratifying facts by $P_{\text{clean}}(o)$ leaves the location unchanged while only its magnitude grows with probability, and this pattern holds across all models—the ARMs simply place more

mass at high probability (Appendix C). What is invariant across paradigms is thus not the magnitude but the *location* of recall.

In the early layers of the last subject token, Qwen and Dream show a clear attention contribution that is largely absent in LLaMA and LLaDA. For Dream this seems consistent with its initialization from Qwen. Either way, this concerns a secondary attention signal; the primary recall site is shared by all four models.

3.3 Editing Facts

Causal tracing (§3.2) reveals where a fact is causally localized; this section asks where editing is actually *effective*. The two need not coincide—the most causally implicated location is not necessarily the most effective one to edit (Hase et al., 2023)—so rather than assuming the tracing result, we measure editing performance directly. We report three analyses: how performance varies with the edited *token position*, with the edited *layer*, and how robust the edits are under generation settings that reflect the MDM's decoding process.

Transferring locate-then-edit to MDMs. We keep the locate-then-edit algorithm (MEMIT) unchanged and apply only the minimal adjustments needed to match the inference behavior of an MDM. Both paradigms optimize the same target value (Eq. 1); they differ only in the conditioning state under which o^* is scored.

The ARM appends the answer tokens directly to the rewriting prompt and reads o^* as the next-token continuation, as it is trained to do. An MDM instead recovers masked positions. We place a block of mask tokens at the answer slot of each rewriting prompt, one for each token of the edit target o^*, and train the edit so that the model reconstructs o^* there. For instance, to edit the team Messi plays for to "Inter Miami", the answer slot holds two mask tokens, and the edit is scored on whether the model reconstructs "Inter" and "Miami" jointly. Because an MDM uses bidirectional attention, the mask tokens at the object positions can influence the fact-lookup hidden state even though they appear after the lookup position in the input. We therefore compute both the edit target and the key vector (§2.1) using the same masked-input format the model sees at inference. The full procedure is in Appendix D.

Token position. We first vary the token position at which the edit is applied—the first and last subject tokens, the first token after the subject, and the first answer position—and measure editing performance (Figure 2). The first answer position is the last prompt token for an ARM and the first appended mask token for an MDM. In both MDMs, editing peaks at the *last subject token* and stays high at the first, then drops sharply for positions after the subject. This matches the recall site that Meng et al. (2023a) report for ARMs and that our causal tracing finds in §3.2, and it holds for both MDMs despite their bidirectional attention. ARM results are in Appendix F.

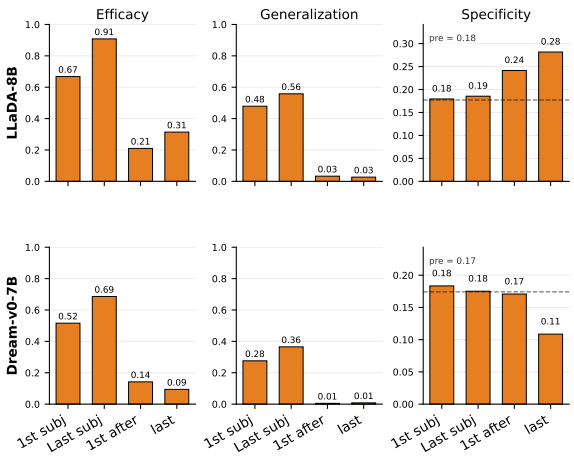

Figure 2: Token position ablation for editing.

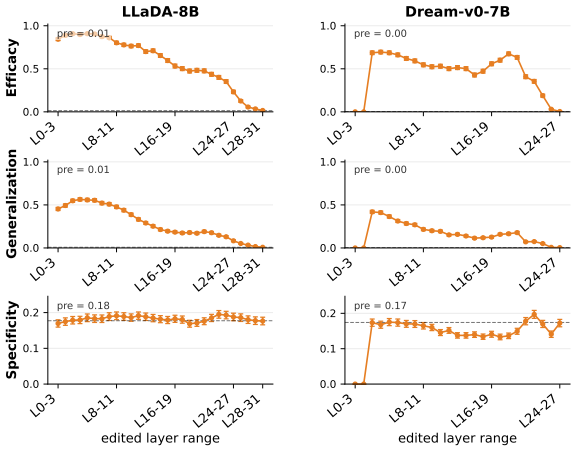

Figure 3: Layer ablation for editing.

Layer. We next vary the range of edited layers and measure editing performance (Figure 3; ARM results in Appendix F). In both MDMs, editing succeeds over a broad range of layers, and taking efficacy and generalization together, the best-performing layers lie in the early-to-mid range, in line with prior work (Meng et al., 2023a). This range overlaps the causal-tracing site but does

	LLaDA			Dream		
Length / steps	Eff.	Gen.	Spec.	Eff.	Gen.	Spec.
pre-edit	0.01	0.01	0.18	0.00	0.00	0.17
target-length	0.91	0.56	0.19	0.95	0.56	0.22
8 / 8	0.82	0.45	0.28	0.92	0.58	0.26
16 / 16	0.86	0.51	0.27	0.88	0.55	0.31
32 / 32	0.84	0.50	0.25	0.91	0.58	0.34

Table 1: Evaluation on COUNTERFACT (mostly single-token targets) under varied generation settings, reported as (generation length)/(denoising steps). *Target-length* matches the generation length to the target; *pre-edit* is the unedited model measured under that setting.

	COUNTERFACT		KAMEL	
Target length	LLaDA	LLaMA	LLaDA	LLaMA
1 token	20,442	20,534	8,815	8,686
2 tokens	286	343	12,055	11,912
3 tokens	149	0	11,187	11,072
4+ tokens	0	0	21,230	21,617
Total	20,877	20,877	53,287	53,287

Table 2: Distribution of target token lengths under each model's tokenizer. Results for Dream and Qwen are in Appendix H.

N	LLaMA	LLaDA	Qwen	Dream
1	0.975	0.915	1.000	0.970
2	0.950	0.595	0.985	0.665
3	0.915	0.325	0.965	0.455
4	0.885	0.265	0.950	0.375

Table 3: Editing efficacy on KAMEL by target token length N (n=200 facts per pair).

not coincide with it exactly, as also reported in prior work (Hase et al., 2023); we therefore fix each model's edit layer by direct measurement rather than from tracing (selection detailed in Appendix E).

Robustness to generation settings. Unlike an ARM, which decodes one token at a time from left to right, an MDM fills a fully masked output over a number of denoising steps, so its generation depends on both the generation length and the number of steps. Since the edit is optimized once at the target length (§3.3), a natural concern is whether it holds under other settings. We therefore take the edited model as is and vary the generation length and step count over $\{8, 16, 32\}$ at evaluation (Table 1). Performance is stable: efficacy stays near 0.85 for LLaDA and 0.90 for Dream across all settings, with generalization and specificity similarly steady. The edit thus holds across decoding configurations it was not tuned for.

4 What Breaks: Editing Multi-Token Facts

The previous section showed that locate-then-edit transfers across paradigms: the MDM localizes facts to the same site as the ARMs, and editing succeeds there. That success, however, was measured on COUNTERFACT, whose targets are almost always a single token. The edit is optimized for one state—the fully masked answer. A single-token target is unmasked in one step from exactly this state, so the edit applies as intended. A multi-token target is unmasked over several steps, each conditioned on the tokens revealed so far—intermediate states the edit was never optimized for. An ARM avoids this, decoding left to right just as its edit is optimized, so multi-token targets pose no such mismatch. It is here, in the MDM's iterative generation, that editing breaks.

In this section we show how this failure grows with target length (§4.1), diagnose its cause (§4.2), and show that a simple correction restores performance (§4.3).

4.1 Editing Degrades with Target Length

The editing results of §3.3 were measured on COUNTERFACT, whose targets are overwhelmingly single-token (Table 2). Inspecting where the MDM fails there, we already find its failures falling at a sharply higher rate on multi-token targets than on single-token ones, even though such targets make up only a few percent of the data (Appendix G). This points to target length as the relevant factor, but COUNTERFACT contains too few multi-token targets to support a systematic analysis.

To study this across a controlled range of target lengths, we turn to KAMEL (Kalo and Fichtel, 2022), a Wikidata-based knowledge benchmark whose entities span single and multiple tokens. Built for probing rather than editing, it is adapted into an editing evaluation with targets grouped by token length (Appendix H). This varies a different axis from the robustness study in §3.3: there we held the target fixed and varied the output budget; here we vary the number of tokens the target itself occupies.

The two MDMs and the two ARMs separate cleanly (Table 3). At $N = 1$ all four models edit well, and the ARMs stay high as N grows—LLaMA from 0.98 to 0.89, Qwen near-

Target	Generated	Pattern
trance = [tr, ance]	*dance* = [d, ance]	2nd ok, 1st wrong
trance = [tr, ance]	*trit* = [tr, it]	1st ok, 2nd wrong

Table 4: Representative multi-token edit failures in the MDM. Individual tokens are often locally plausible, but they fail to combine into the intended target.

N	#Facts	Token-level	Full target	Gap
2	200	73.5%	59.5%	+14.0
3	200	53.8%	32.5%	+21.3
4	200	50.7%	26.5%	+24.2

Table 5: Token-level vs. joint recovery for LLaDA on multi-token KAMEL targets. *Token-level* is the rate at which a target token appears anywhere in the generation; *full target* is the rate at which the complete target is produced (efficacy). The gap between them widens with target length N.

		LLaDA		Dream	
N	Metric	MEMIT†	+ Correction	MEMIT†	+ Correction
2	Eff.	0.60	0.87	0.67	0.88
	Gen.	0.36	0.52	0.46	0.81
3	Eff.	0.33	0.76	0.46	0.59
	Gen.	0.20	0.36	0.32	0.56
4	Eff.	0.27	0.73	0.38	0.51
	Gen.	0.14	0.36	0.25	0.39

Table 6: Editing performance on KAMEL for multi-token targets (n=200 per N). †MEMIT denotes our MDM adaptation of MEMIT (§3.3); + *correction* fits the edit across partially unmasked states rather than the fully masked one alone.

perfect throughout. The MDMs instead fall steeply: LLaDA from 0.92 at $N = 1$ to 0.27 at $N = 4$, and Dream from 0.97 to 0.37. Within each ARM–MDM pair the two models are evaluated on the same facts at each length (Appendix H), so the difference that opens up reflects target length rather than the choice of facts. This gap, invisible for single-token edits, grows steadily with the target.

4.2 The Edit Is Not Optimized for Intermediate States

To understand where this failure originates, we examine whether the model fails because it does not know the right tokens of the target object or because it cannot assemble them in the right positions. We measure two rates: how often each target token (i.e., each token in the target object) appears *anywhere* in the generation, and how often the *complete* target is produced (Table 5). The first is much higher than the second—roughly half the target tokens surface somewhere, yet far fewer full targets are generated eventually. That is, the model has the right tokens; it just does not combine them into the target. Example failure cases show this directly: editing "trance" as ["tr", "ance"], the model produces "ance" but misses the first token, giving "dance", or gets the first and misses the rest (Table 4). And this gap—between having the tokens and assembling them—widens with target length, from an absolute difference of 14% at N=2 to 24% at N=4.

This points to a mismatch between the state the edit is optimized for and the states it actually operates on. As described in §2.1, MEMIT optimizes a single residual $\boldsymbol{\delta}$ so that the model produces the target from one state: the fully masked answer. But an MDM generates a multi-token target over several steps rather than from that single state (§2.2). It unmasks one position at a time, passing through intermediate states in which some positions are already filled while others remain masked—states $\boldsymbol{\delta}$ was never optimized for. And because the MDM is bidirectional, each position it unmasks becomes context that reshapes the predictions at the positions still masked. A single token can therefore be recovered in isolation, which is why the right tokens do appear; but the moment one is unmasked, it shifts the conditions for the rest, under which $\boldsymbol{\delta}$ was never fit. Over a multi-token target these shifts compound—the more positions, the longer the chain and the wider the gap we observe.

This explains our earlier observations. A single-token target has only one position and no intermediate state, so the edit operates exactly where it was optimized—which is why single-token edits were comparable for both ARMs and MDMs. Longer targets traverse more intermediate states, where failures accumulate. The ARM never faces this mismatch, as it generates left to right in one pass. The failure thus stems not from *where* the edit is applied but from *how* the edited fact is generated, and is specific to the iterative denoising of MDMs.

4.3 A Correction for Intermediate States

The residual $\boldsymbol{\delta}$ (Eq. 1) is optimized only on the fully masked state, yet inference visits that state once and spends the rest of generation on partially unmasked ones. If this is the cause, the remedy is to fit $\boldsymbol{\delta}$ on those intermediate states as well. We

broaden only the inputs on which $\boldsymbol{\delta}$ is fit, leaving the update rule and edit layers as they are. During optimization we vary how many of the N target tokens are already unmasked, cycling through every count from none to $N-1$, and at each step we resample which positions are revealed. Sweeping the count this way exposes $\boldsymbol{\delta}$ to the same progression generation runs through, from the fully masked answer, through each partially unmasked stage, to the verge of completion, rather than to the masked state alone. It thus learns to recover the target from each state inference may encounter. Resampling the revealed positions, rather than fixing an order, keeps $\boldsymbol{\delta}$ from overfitting to one unmasking pattern; schedules and ablations are in Appendix I.

This single change substantially fills the multi-token cliff (Table 6). Both efficacy and generalization rise across every target length and for both models—LLaDA's efficacy at $N{=}4$ roughly triples, from 0.27 to 0.73, and Dream's generalization at $N{=}2$ climbs from 0.46 to 0.81. Everything else in the locate-then-edit procedure of §2.1 is unchanged—the closed-form update, the edited layers, the key and target value—and only the conditioning states in Eq. 1 are broadened from the single masked state to include the intermediate ones. That so narrow a change recovers a large part of the drop suggests the intermediate-state mismatch is an important factor, consistent with our diagnosis.

5 Related Work

Knowledge Editing. Knowledge editing methods fall into three paradigms. *Memory-based approaches* store edited knowledge externally and retrieve it at inference, leaving the model untouched (Mitchell et al., 2022a; Hartvigsen et al., 2023). *Meta-learning approaches* train auxiliary networks to predict the weight updates (De Cao et al., 2021; Mitchell et al., 2022a). *Locate-then-edit approaches* instead identify the parameters associated with a fact and modify them directly (Meng et al., 2023a,b). This last family has been especially influential: by grounding each edit in a causal analysis of where a fact is localized, it makes the edit site interpretable, and has since scaled to many simultaneous edits while reducing side effects (Fang et al., 2025; Li et al., 2024; Park et al., 2026; Gupta et al., 2024). We take it as our starting point precisely because its edit location is derived from an explicit analysis of the model—the natural setting in which to ask whether such an analysis, developed on ARMs, carries over to MDMs.

ARMs vs. MDMs. A growing body of work compares the two paradigms. Some studies contrast their capabilities, finding advantages of MDMs in knowledge acquisition (Pan et al., 2026) or differences in long-context behavior (Liu et al., 2025). Closer to our setting, Piskorz et al. (2025) show that appended mask tokens distract context comprehension and propose a mask-agnostic loss, and Wang et al. (2026) interpret MDM representations through sparse features. These analyses operate on features or inference-time behavior; none addresses where a fact resides in the weights or how to edit it, and where they modify training objectives or activations, we leave the model's training untouched and intervene only on the edit. To our knowledge, ours is the first work to bring locate-then-edit knowledge editing, developed on ARMs, to MDMs, and to compare the two paradigms from this perspective.

6 Conclusion

We transferred locate-then-edit knowledge editing from ARMs to MDMs. The two agree on *where* an edit applies: facts concentrate at the early-to-mid-layer MLP of the last subject token in both, and editing there matches the ARM on standard benchmarks. They diverge on *how* the edited fact is generated. Because such benchmarks are almost entirely single-token, this divergence is invisible on them; evaluating across target lengths reveals that MDM editing degrades as the target grows, since iterative denoising visits intermediate states the edit was never optimized for. Including those states in the optimization restores most of the lost performance, confirming the cause; a gap at the longest targets remains for future work. Editing an MDM thus turns not only on where a fact is placed but on how it is generated—porting an editing method across architectures means matching the generation process, not only the site it acts on.

Limitations

We instantiate locate-then-edit with MEMIT, the most established method of its kind. Other methods in this family share the same locate-and-edit structure and could plausibly transfer to MDMs in the same way, though we have not verified this. Our analysis also covers two ARM–MDM pairs at a single scale. We chose these pairs because each

matches an MDM to an ARM of the same architecture and size, which isolates the effect of the training objective—the cleanest such pairings currently available; whether the findings hold at larger scales or for other MDM families remains open. Finally, we identify and address one cause of multi-token failure—the mismatch between the single state the edit is optimized for and the intermediate states generation visits. This accounts for most of the degradation but not all of it: our correction exposes the edit to how many positions are unmasked, but not to which positions or in what order, and a gap remains at the longest targets. Closing it may require accounting for the order in which an MDM unmasks its output, which we leave to future work.

Ethics Statement

Knowledge editing modifies the facts a model produces, and the same techniques that correct outdated or erroneous knowledge can in principle be used to insert false or misleading information. Our work studies whether and how locate-then-edit transfers to masked diffusion models, rather than proposing editing for deployment, and our experiments use standard public benchmarks (COUNTERFACT, KAMEL) and openly available models. We nonetheless note that editing methods should be applied with appropriate safeguards—such as provenance tracking and human oversight—particularly as they extend to new model families. We see no additional risks specific to our analysis beyond those already present in the knowledge-editing literature.

References

James A. Anderson. 1972. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14(3):197–220.

Louis Bethune, David Grangier, Dan Busbridge, Eleonora Gualdoni, Marco Cuturi, and Pierre Ablin. 2025. Scaling laws for forgetting during fine-tuning with pretraining data injection. *Preprint*, arXiv:2502.06042.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6491–6506, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. 2025. Alphaedit: Null-space constrained model editing for language models. In *The Thirteenth International Conference on Learning Representations*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 31 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Akshat Gupta, Dev Sajnani, and Gopala Anumanchipalli. 2024. A unified framework for model editing. *Preprint*, arXiv:2403.14236.

Thomas Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. 2023. Aging with grace: Lifelong model editing with discrete key-value adaptors. *Preprint*, arXiv:2211.11031.

Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun. 2023. Does localization inform editing? surprising differences in causality-based localization vs. knowledge editing in language models. *Preprint*, arXiv:2301.04213.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun KIM, Stanley Jungkyu Choi, and Minjoon Seo. 2022. Towards continual knowledge learning of language models. In *International Conference on Learning Representations*.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12).

Jan-Christoph Kalo and Leandra Fichtel. 2022. Kamel: Knowledge analysis with multitoken entities in language models. In *Automated Knowledge Base Construction*.

Teuvo Kohonen. 1972. Correlation matrix memories. *IEEE Transactions on Computers*, C-21(4):353–359.

Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. 2024. Pmet: Precise model editing in a transformer. *Preprint*, arXiv:2308.08742.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.

Xiaoran Liu, Yuerong Song, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025. Longllada: Unlocking long context capabilities in diffusion llms. *Preprint*, arXiv:2506.14429.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *Preprint*, arXiv:2308.08747.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2023a. Locating and editing factual associations in gpt. *Preprint*, arXiv:2202.05262.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. 2023b. Mass editing memory in a transformer. *The Eleventh International Conference on Learning Representations (ICLR)*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. 2022a. Fast model editing at scale. In *International Conference on Learning Representations*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. 2022b. Memory-based model editing at scale. *Preprint*, arXiv:2206.06520.

Shen Nie, 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. *Preprint*, arXiv:2502.09992.

Xu Pan, Ely Hahami, Jingxuan Fan, Ziqian Xie, and Haim Sompolinsky. 2026. Diffusion-inspired masked fine-tuning for knowledge injection in autoregressive llms. *Preprint*, arXiv:2510.09885.

Haewon Park, Sangwoo Kim, and Yohan Jo. 2026. SUIT: Knowledge editing with subspace-aware key-value mappings. In *The Fourteenth International Conference on Learning Representations*.

Julianna Piskorz, Cristina Pinneri, Alvaro Correia, Motasem Alfarra, Risheek Garrepalli, and Christos Louizos. 2025. Masks can be distracting: On context comprehension in diffusion language models. *Preprint*, arXiv:2511.21338.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Peng Wang, Ningyu Zhang, Bozhong Tian, Zekun Xi, Yunzhi Yao, Ziwen Xu, Mengru Wang, Shengyu Mao, Xiaohan Wang, Siyuan Cheng, Kangwei Liu, Yuansheng Ni, Guozhou Zheng, and Huajun Chen. 2024a. EasyEdit: An easy-to-use knowledge editing framework for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 82–93, Bangkok, Thailand. Association for Computational Linguistics.

Song Wang, Yaochen Zhu, Haochen Liu, Zaiyi Zheng, Chen Chen, and Jundong Li. 2024b. Knowledge editing for large language models: A survey. *Preprint*, arXiv:2310.16218.

Xu Wang, Bingqing Jiang, Yu Wan, Baosong Yang, Lingpeng Kong, and Difan Zou. 2026. Dlm-scope: Mechanistic interpretability of diffusion language models via sparse autoencoders. *Preprint*, arXiv:2602.05859.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Editing large language models: Problems, methods, and opportunities. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10222–10240, Singapore. Association for Computational Linguistics.

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

Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. 2020. Modifying memories in transformer models. *Preprint*, arXiv:2012.00363.

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, Hongrui Guo, Jiaqi Hu, Wentao Ye, Tieyuan Chen, Chenchen Li, Chengfu Tang, Haibo Feng, Jun Hu, Jun Zhou, and 7 others. 2025. Llada-moe: A sparse moe diffusion language model. *Preprint*, arXiv:2509.24389.

A MEMIT Closed-Form Update

This appendix expands the closed-form update summarized in §2.1, following Meng et al. (2023b). We use the notation of §2.1 throughout: $\mathbf{W}$ is the MLP down-projection viewed as a linear associative memory, $\mathbf{k}$ a key, and $\mathbf{v}$ its value.

Single-layer associative memory. A layer storing a set of key–value pairs $\{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{n}$ holds the weights that minimize the squared recall error,

$$\mathbf{W} = \arg\min_{\widehat{\mathbf{W}}} \sum_{i=1}^{n} \left\| \widehat{\mathbf{W}}\mathbf{k}_i - \mathbf{v}_i \right\|^2, \tag{3}$$

which, stacking the keys and values into matrices $\mathbf{K} = [\mathbf{k}_1 \mid \cdots \mid \mathbf{k}_n]$ and $\mathbf{V} = [\mathbf{v}_1 \mid \cdots \mid \mathbf{v}_n]$, is solved by the normal equation $\mathbf{W}\mathbf{K}\mathbf{K}^\top = \mathbf{V}\mathbf{K}^\top$.

Adding a new association. An edit asks the same weights to additionally map a set of new keys $\mathbf{K}_1$ to target values collected in $\mathbf{V}_1$, the matrix whose columns are the $\mathbf{v}^*$ of §2.1, while preserving the existing associations. Writing the updated weights as $\mathbf{W} + \mathbf{\Delta}$ and solving the corresponding normal equation in block form yields a closed-form update (Meng et al., 2023b):

$$\mathbf{\Delta} = \mathbf{R}\,\mathbf{K}_1^\top \left(\mathbf{C} + \mathbf{K}_1\mathbf{K}_1^\top\right)^{-1}, \tag{4}$$

where $\mathbf{R} = \mathbf{V}_1 - \mathbf{W}\mathbf{K}_1$ is the residual of the new targets under the current weights, and $\mathbf{C}$ is the uncentered covariance of the pre-existing keys. Since the original keys are not accessible, $\mathbf{C}$ is estimated as $\lambda\,\mathbb{E}[\mathbf{k}\mathbf{k}^\top]$ over a sample of inputs, with λ balancing the new associations against the preserved ones.

Optimizing the target value. The target value $\mathbf{v}^* = \mathbf{v} + \boldsymbol{\delta}$ enters Eq. 4 through $\mathbf{V}_1$, and its residual $\boldsymbol{\delta}$ is the quantity optimized in Eq. 1: it is chosen so that writing $\mathbf{v}^*$ at the edit location drives the model to produce the target object o^*. To make the edit robust across contexts, the objective averages over prompts formed by prepending random prefixes to the rewriting prompt p. For an ARM these prompts place o^* as the next-token continuation; for an MDM we instead append L `[MASK]` tokens and score o^* at the masked positions, as described in §3.3.

Spreading the update over layers. Rather than editing a single layer, MEMIT distributes the update over the contiguous range $\mathcal{R}$ of critical MLP layers identified by causal tracing. The full residual is computed at the last layer $L = \max(\mathcal{R})$, and each layer $l \in \mathcal{R}$ absorbs a fraction of it, scaled by $1/(L - l + 1)$ so that the change is shared evenly across the range. Because editing one layer shifts the activations of later ones, the keys $\mathbf{K}_l$ are recomputed and the layers are updated in ascending order.

B Evaluation Details

We adopt the three standard knowledge-editing criteria (Meng et al., 2023a,b; Fang et al., 2025), but assess all of them from the model's *generation* rather than from probability comparisons, so that the same protocol applies uniformly to ARMs and MDMs. For each item i, let x_i be the rewrite prompt (s_i, r_i), $N(x_i)$ its paraphrase prompts, and $O(x_i)$ its neighborhood prompts (a distinct subject s_i' sharing the relation r_i and object o_i). Let o^* denote the new target object and o the original object.

Generation protocol. Given a prompt, the model generates a continuation of length $T = \max\big(|\tau(\sqcup o^*)|, |\tau(\sqcup o)|\big)$, where $\tau(\cdot)$ denotes tokenization and $\sqcup$ a leading space. A single generation of this length is scored against both objects, which lets us track not only whether the new target o^* appears but also whether the original o still does. For ARMs we use greedy left-to-right decoding of T tokens. For MDMs we run denoising over a fully masked suffix, committing one position per step in descending order of model confidence, with the number of steps equal to the suffix length. LLaDA uses a suffix of length T; for Dream we extend it by one mask slot to leave room for its natural generation pattern, such as an EOS marker or a function word, without which generation degrades to copying the first token or terminating early.

Scoring. An edit *hits* a target t on a prompt p if t appears as a case-insensitive substring of the generated text (after whitespace stripping):

$$\mathrm{hit}(p, t) = \mathbf{1}\big[\, t \subseteq_{\text{substr}} \mathrm{gen}(p) \,\big]. \tag{5}$$

The three criteria are then

$$\text{Efficacy} = \mathbb{E}_i\, \mathrm{hit}(x_i, o^*), \tag{6}$$

$$\text{Generalization} = \mathbb{E}_i\, \mathbb{E}_{p \in N(x_i)}\, \mathrm{hit}(p, o^*), \tag{7}$$

$$\text{Specificity} = \mathbb{E}_i\, \mathbb{E}_{p \in O(x_i)}\, \mathrm{hit}(p, o), \tag{8}$$

where efficacy and generalization check that the new target o^* is produced on the rewrite and paraphrase prompts, while specificity checks that the original object o is *preserved* on neighborhood prompts.

C Causal Tracing Details

This appendix details the causal tracing setup summarized in §3.2, following the procedure of Meng et al. (2023a).

Dataset and fact selection. Tracing is run on the KNOWNS dataset of Meng et al. (2023a) (1,209 facts), separate from the COUNTERFACT and KAMEL sets used for editing evaluation. Following Meng et al. (2023a), we restrict the analysis to facts that the clean model already predicts correctly, i.e., whose first decoded token of o matches the full answer string. Since this filter depends on each model's own predictions, the retained subset differs across the four models.

C.1 MDM Adaptation

Because an MDM predicts at mask positions rather than at a next token, we append a block of mask tokens to the prompt and read the score at the first of them. The score $P(o)$ is the marginal probability of the first token of o at this position, taken as the clean run's greedy token. In practice the correctness filter above requires this first token, once decoded, to match the full answer string, so the trace is restricted to single-token answers; this matches the ARM case, where $P(o)$ is the first-token probability at the last prompt position. We use a block of 8 mask tokens throughout; since only single-token answers are retained, 8 positions are more than sufficient and the answer token is always scored at the first mask position.

Tracing uses a single forward pass over the masked input rather than the iterative denoising used at evaluation. A single pass exposes each layer's contribution directly, which is what layer-by-layer restoration requires; iterative denoising would instead entangle contributions across steps.

The corruption is applied exactly as in the ARM case: Gaussian noise is added to the embeddings of the subject tokens only, leaving the appended mask block untouched. The noise level $\nu = 3\sigma$ is calibrated to each model, with σ the standard deviation of subject-token embeddings, estimated separately per model from its own embeddings.

Model	Mean	Median	t_1	t_2
LLaMA-3-8B	0.71	0.78	0.60	0.91
LLaDA-8B	0.54	0.56	0.41	0.70
Qwen2.5-7B	0.49	0.46	0.35	0.59
Dream-v0-7B	0.18	0.16	0.14	0.21

Table 7: Distribution of $P_{\text{clean}}(o)$, the clean-run first-token probability of the correct object, over the traced facts of each model. The tertile boundaries t_1 (lower) and t_2 (upper) are the cut points used for the stratified analysis.

C.2 Tracing Window

When tracing the full hidden state we restore a single state $h_{\hat{i}}^{(\hat{l})}$. When tracing a sub-module (MLP or attention), a single layer's contribution is too small to recover the prediction on its own, so we restore a contiguous window of 10 layers centered at $\hat{l}$ at token $\hat{i}$, following Meng et al. (2023a).

C.3 Normalized AIE

We report the *normalized AIE*, the standard fractional tracing effect of Meng et al. (2023a): the recovery a restored state provides as a fraction of the gap that corruption induces,

$$\text{normAIE}(\hat{i}, \hat{l}) = \mathbb{E}_{\text{facts}}\left[\frac{P_{\text{restored}}(o) - P_{\text{corrupted}}(o)}{P_{\text{clean}}(o) - P_{\text{corrupted}}(o)}\right] \times 100\% \tag{9}$$

where P_{clean}, $P_{\text{corrupted}}$, and P_{restored} are the first-token probabilities of o under the clean, corrupted, and corrupted-with-restoration runs. A value of 0 means no recovery beyond the corrupted run and 100% a full recovery. Reporting it as a percentage places the four models on a common scale despite their different baseline probabilities. Facts whose corruption gap $P_{\text{clean}}(o) - P_{\text{corrupted}}(o)$ is negligible are excluded, since the ratio is then undefined.

Stratified analysis. The MDMs show a weaker average effect than the ARMs. To check that this reflects their probability distribution rather than weaker localization, we partition each model's traced facts into tertiles by $P_{\text{clean}}(o)$ (Table 7) and recompute the normalized AIE within each. Across all tertiles the effect peaks at the same location—the early-to-mid layer at the last subject token—while only its magnitude grows with $P_{\text{clean}}(o)$. The ARMs simply place more mass at high probability, so their stronger average reflects this distribution, not sharper localization.

D Locate-then-Edit on MDMs

This appendix expands the transfer described in §3.3 to the level of the implementation. The closed-form solver, the key/value framework, and the delta optimization loop of MEMIT (Meng et al., 2023b) are kept intact; the only change is that every input distribution on which MEMIT runs a forward pass is mask-augmented to match the bidirectional attention of an MDM.

A single MEMIT edit runs forward passes on four distinct input distributions. In the ARM case all of them use the raw prompt (or prompt followed by the target); in the MDM case we put all of them in the same form, a prompt followed by a block of mask tokens at the answer slot, with one mask token per token of the edit target o^*.

1. **Rewriting prompts.** The main term of Eq. 1: we append the mask block to each fact's rewriting prompt and supervise the logits at the mask positions toward o^*. The ARM's next-token logits at the final answer positions thus map onto the logits read at the mask positions.

2. **KL-anchor prompt.** Beyond the term shown in Eq. 1, the optimization also includes a KL regularizer that keeps unrelated predictions stable. Its anchor is likewise mask-augmented: in place of the standard subject-last next-token logits, we read the logits at the mask position of the anchor prompt followed by a mask token.

3. **Key extraction.** The key $\mathbf{k}$ for each fact is read as the subject-last hidden state on the mask-augmented prompt.

4. **Residual baseline.** The baseline value used to form the closed-form residual is likewise read from the subject-last hidden state on the mask-augmented prompt.

Under the causal attention of an ARM, the subject-last hidden state is identical on the raw prompt and on the prompt followed by the target, since future tokens cannot be attended to; the two hidden states that MEMIT reads—the optimization target and the residual baseline—then coincide automatically. Under the bidirectional attention of an MDM this coincidence breaks, so we force all four distributions to be defined over the same masked context.

Hyperparameters. The two ARMs follow the MEMIT settings provided in EasyEdit (Wang et al., 2024a): learning rate 10^{-1}, 25 gradient steps for the target optimization, clamp-norm factor 0.75, and KL factor 0.0625. For the two MDMs, we inherit the hyperparameters of their architectural sibling without retuning: LLaDA from LLaMA (both share the LLaMA-style transformer block) and Dream from Qwen (Dream's backbone is Qwen2.5). The closed-form solver, the covariance term, the weight update, and the distribution of the residual across layers are all as in the original MEMIT (Meng et al., 2023b). The only setting we determine per model is the *edit layer*, chosen by a direct layer sweep that selects the best 4-layer window by the rank-sum of efficacy and generalization; this yields LLaMA-3 L1–4, LLaDA L4–7, Qwen2.5 L7–10, and Dream L3–6 (Appendix E).

E Editing Layer Selection

The edit layer is the one setting we tune per model, since editing performance is sensitive to which MLP layers are modified and prior defaults do not transfer reliably across our four models. For each model we sweep every 4-layer contiguous window on a held-out subset of COUNTERFACT (500 facts), score each window by the rank-sum of its efficacy and generalization ranks, and select the best-scoring window (ties broken by efficacy). We exclude specificity from the criterion because it trades off against efficacy: the highest-specificity windows tend to under-edit. The sweep uses the same evaluation settings as the main experiments (Appendix B) and the hyperparameters of Appendix D.

F ARM Editing Results

For completeness, we report the ARM counterparts of the editing analyses in the main text. Figure 4 shows the layer sweep for the two ARMs, and Figure 5 shows the token position ablation. The trends mirror those of the MDMs in §3.3.

G Failure Analysis on COUNTERFACT by Target Length

The controlled KAMEL study in §4.1 was motivated by a length trend already visible on COUNTERFACT. Here we report the per-length editing results on the COUNTERFACT sweep set ($n = 500$), with each model evaluated at its selected edit layer (Appendix E). Because COUNTERFACT is single-token

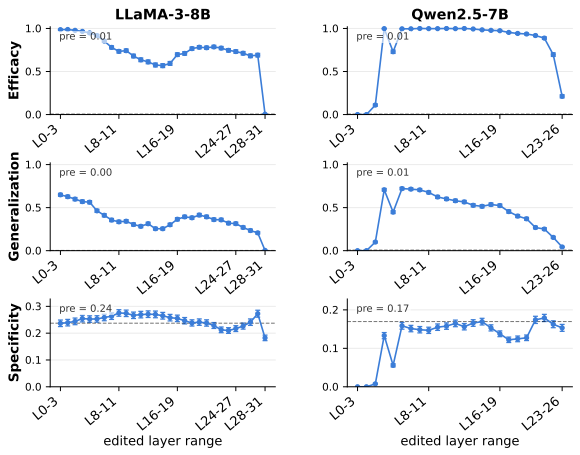

Figure 4: Layer sweep for ARM editing.

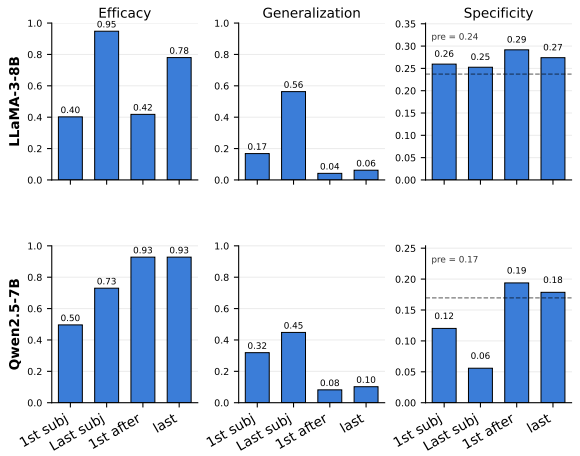

Figure 5: Token position ablation for ARM editing.

dominated, the multi-token cells contain few facts; we report them to show the trend, not as precise estimates.

The single-token gap between each MDM and the ARM is modest (∼3.5 pp for Dream, ∼6.9 pp for LLaDA), but it widens sharply with target length: at $N = 2$ it grows to ∼23 pp for Dream and ∼34 pp for LLaDA, while LLaMA itself loses only ∼7 pp from $N = 1$ to $N = 2$. The same effect appears on KAMEL under controlled length groups (§4.1), where the larger per-length samples make the trend precise; the agreement between the two datasets indicates it is not an artifact of COUNTERFACT.

H Curating KAMEL for Multi-Token Editing

We build the multi-token editing set from KAMEL (Kalo and Fichtel, 2022), recast into the COUNTERFACT schema so it runs through the same evaluation pipeline (§3.3). To trace editing across target length we need facts at each length N, which COUNTERFACT does not provide; KAMEL spans a wide range of entity lengths. We cover $N \in \{1, 2, 3, 4, 5\}$ and keep a fact at length N only if its object tokenizes to exactly N tokens under *both* the LLaDA and LLaMA tokenizers, so that any ARM–MDM difference reflects target length rather than one tokenizer splitting the same target more finely than the other.

Target length	LLaDA	LLaMA	Dream
$N = 1$	92.1 (443/481)	99.0 (482/487)	95.5 (465/487)
$N = 2$	58.3 (7/12)	92.3 (12/13)	69.2 (9/13)
$N = 3$	57.1 (4/7)	—	—
Overall	90.8 (454/500)	98.8 (494/500)	94.8 (474/500)

Table 8: Editing efficacy (%) on the COUNTERFACT sweep set ($n = 500$), broken down by target token length under each model's tokenizer. Counts are successes/total; dashes mark lengths absent under that tokenizer. Both MDMs (LLaDA, Dream) drop sharply on multi-token targets ($N \geq 2$), with failure rates several times that at $N = 1$, whereas the ARM (LLaMA) loses only ∼7 pp over the same step.

	COUNTERFACT		KAMEL	
Target length	Qwen	Dream	Qwen	Dream
1 token	20,534	20,534	8,682	8,682
2 tokens	343	343	11,834	11,834
3 tokens	0	0	10,896	10,896
4+ tokens	0	0	21,875	21,875
Total	20,877	20,877	53,287	53,287

Table 9: Distribution of target token lengths under Qwen and Dream tokenizers (extension of Table 2). The two columns within each dataset are identical because Dream is built on the Qwen2.5 backbone and shares its tokenizer; both confirm the same overall picture observed under LLaMA and LLaDA — CounterFact is dominated by single-token targets with no facts at $N \geq 4$, while KAMEL spans the full range with the majority of facts at $N \geq 2$.

Construction. Starting from the single-answer facts in KAMEL (54 relations, one cloze template each), we measure object token length in context as in our evaluation (Appendix B) and apply the two-tokenizer filter above. For each fact we form a counterfactual target by swapping in another object from the same relation that is also N tokens under both tokenizers, which keeps the mask budget fixed at N and the substitution semantically plausible. Since KAMEL provides only one template

per relation, we write one paraphrase per relation to measure generalization. From the facts passing all filters we sample 200 per N.

N	1	2	3	4
Samples	200	200	200	200
Relations covered	18	34	35	34

Table 10: KAMEL editing set by target length. The lower relation count at N=1 reflects that objects that are single tokens under both tokenizers concentrate in a few relations (e.g., short country names); $N \geq 2$ draws on over thirty relations.

I Partial-Mask Augmentation: Details and Ablations

This appendix specifies the partial-mask augmentation of §4.3 and ablates its two design choices: how often each partially unmasked state is visited during optimization, and how the revealed positions are chosen.

I.1 Method

The correction broadens the states on which the residual $\boldsymbol{\delta}$ (Eq. 1) is fit, leaving everything else in MEMIT unchanged. For a target of N tokens, at optimization step t we reveal $k = t \bmod N$ of the target positions, cycling through $k = 0, 1, \ldots, N-1$ over training. At each step we pick k positions, fill them with the corresponding target tokens, mask the rest, and update $\boldsymbol{\delta}$ on the $N-k$ still-masked positions. The fully masked state (k=0) and every partially unmasked state thus appear in turn, matching the progression generation runs through.

I.2 Ablation 1: Which states to visit

We fix everything except how the count k is distributed across optimization, varying only which partially unmasked states are emphasized (Table 11, LLaDA on KAMEL).

The largest gain comes simply from adding augmentation at all: moving from (a) to any augmented variant lifts efficacy by roughly 25–30 percentage points (pp) across lengths. Among the augmented variants, emphasizing the harder, less-revealed states (b) beats emphasizing the easier ones (d), and the gap grows with N. Visiting every count equally, whether at random (c) or in a fixed cycle (e), is best overall, with the deterministic cycle slightly ahead at N=3, 4. We adopt the cycle as our default.

State distribution	N=2	N=3	N=4
(a) fully masked only	0.640	0.440	0.390
(b) bias toward fewer revealed	0.880	0.725	0.690
(c) uniform (random order)	0.860	0.710	0.700
(d) bias toward more revealed	0.845	0.645	0.545
(e) cycle (ours)	0.865	**0.760**	**0.725**

Table 11: State-distribution ablation (LLaDA, KAMEL, n=200). (a) trains only on the fully masked state (no augmentation); (b) and (d) skew the count toward fewer or more revealed positions; (c) draws each count uniformly at random; (e) our deterministic cycle visits every count in turn.

I.3 Ablation 2: How to choose revealed positions

Fixing the cycle above, we compare three ways of choosing which k positions to reveal at each step (Table 12, KAMEL N=3).

	LLaDA		Dream	
Reveal policy	Eff.	Gen.	Eff.	Gen.
left-to-right	0.585	0.275	0.605	0.530
by confidence	0.615	0.340	**0.630**	**0.590**
random (ours)	**0.760**	**0.360**	0.590	0.560

Table 12: Reveal-policy ablation (KAMEL N=3, n=200). *Left-to-right* reveals the first k positions; *by confidence* reveals the k positions the unedited model already scores highest; *random* resamples k positions at each step. Best per column in bold.

The two models prefer different policies. For LLaDA, random reveal is clearly best, improving efficacy by over 14 pp on the next-best policy; resampling the revealed set each step appears to make $\boldsymbol{\delta}$ robust to many position combinations rather than to one fixed pattern. For Dream, revealing by confidence is slightly better. We speculate that this reflects its initialization from an ARM (§2.2): with a stronger left-to-right prior, supervising the positions the model is already confident about may align with the order it tends to generate in. Across both models, the fixed left-to-right pattern generalizes worst, suggesting $\boldsymbol{\delta}$ overfits to that specific order. We use random reveal as our default for both models: it is far stronger for LLaDA and competitive for Dream, and a single policy keeps the method uniform across models.

J Computational resources.

All experiments were run on a single NVIDIA H200 GPU (144GB). The models we study have

7–8B parameters (§3.1). Editing time scales with the number of denoising steps and is roughly constant per sample: for LLaDA, a single edit takes about 8.7 seconds per fact (e.g., $\sim$29 minutes for 200 facts), and for Dream about 4.5 seconds per fact ($\sim$15 minutes for 200 facts). A full edit of 500 facts therefore takes approximately 70 minutes for LLaDA and 40 minutes for Dream on one GPU.